Ecole Nationale Supérieure des Mines
SAINT-ETIENNE

N° d'ordre : 2011 EMSE 0630

MÉMOIRE DE THÈSE

présenté par

ROSINE KITIO TEUSSOP

Pour l'obtention du titre de
Docteur de l'École Nationale Supérieure des Mines de Saint-Étienne

Spécialité : Informatique

GESTION DE L'OUVERTURE AU SEIN D'ORGANISATIONS MULTI-AGENTS

UNE APPROCHE BASÉE SUR DES ARTEFACTS ORGANISATIONNELS

Soutenue à Saint-Étienne le 25 Octobre 2011
en présence d'un jury composé de :

Président :	
CATHERINE GARBAY	Professeur, LIG – AMA, Université Joseph Fourier, Grenoble
Rapporteurs :	
MARIE PIERRE GLEIZES	Professeur, IRIT – SMAC, Université Paul Sabatier, Toulouse
JAIME SIMÃO SICHMAN	Professeur, LTI – PCS, Université de São Paulo - Brésil
Examinateurs :	
FLAVIEN BALBO	Maître de conférences, LAMSADE, Université Paris Dauphine, Paris
Co-Encadrant :	
JOMI FRED HÜBNER	Professeur, DAS, Université fédérale de Santa Catarina - Brésil
Directeur de thèse :	
OLIVIER BOISSIER	Professeur, LSTI – Institut Henri Fayol – ISCOD, École Nationale Supérieure des Mines, Saint-Étienne

Spécialités doctorales :

SCIENCES ET GENIE DES MATERIAUX
MECANIQUE ET INGENIERIE
GENIE DES PROCEDES
SCIENCES DE LA TERRE
SCIENCES ET GENIE DE L'ENVIRONNEMENT
MATHEMATIQUES APPLIQUEES
INFORMATIQUE
IMAGE, VISION, SIGNAL
GENIE INDUSTRIEL
MICROELECTRONIQUE

Responsables :

J. DRIVER Directeur de recherche – Centre SMS
A. VAUTRIN Professeur – Centre SMS
G. THOMAS Professeur – Centre SPIN
B. GUY Maître de recherche – Centre SPIN
J. BOURGOIS Professeur – Centre SITE
E. TOUBOUL Ingénieur – Centre G2I
O. BOISSIER Professeur – Centre G2I
JC. PINOLI Professeur – Centre CIS
P. BURLAT Professeur – Centre G2I
Ph. COLLOT Professeur – Centre CMP

Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'État ou d'une HDR)

AVRIL	Stéphane	MA	Mécanique & Ingénierie	CIS
BATTON-HUBERT	Mireille	MA	Sciences & Génie de l'Environnement	SITE
BENABEN	Patrick	PR 1	Sciences & Génie des Matériaux	CMP
BERNACHE-ASSOLANT	Didier	PR 0	Génie des Procédés	CIS
BIGOT	Jean-Pierre	MR	Génie des Procédés	SPIN
BILAL	Essaïd	DR	Sciences de la Terre	SPIN
BOISSIER	Olivier	PR 1	Informatique	G2I
BOUCHER	Xavier	MA	Génie Industriel	G2I
BOUDAREL	Marie-Reine	PR 2	Génie Industriel	DF
BOURGOIS	Jacques	PR 0	Sciences & Génie de l'Environnement	SITE
BRODHAG	Christian	DR	Sciences & Génie de l'Environnement	SITE
BURLAT	Patrick	PR 2	Génie industriel	G2I
COLLOT	Philippe	PR 1	Microélectronique	CMP
COURNIL	Michel	PR 0	Génie des Procédés	SPIN
DAUZERE-PERES	Stéphane	PR 1	Génie industriel	CMP
DARRIEULAT	Michel	IGM	Sciences & Génie des Matériaux	SMS
DECHOMETS	Roland	PR 1	Sciences & Génie de l'Environnement	SITE
DESRAYAUD	Christophe	MA	Mécanique & Ingénierie	SMS
DELAFOSSE	David	PR 1	Sciences & Génie des Matériaux	SMS
DOLGUI	Alexandre	PR 1	Génie Industriel	G2I
DRAPIER	Sylvain	PR 2	Mécanique & Ingénierie	SMS
DRIVER	Julian	DR 0	Sciences & Génie des Matériaux	SMS
FEILLET	Dominique	PR 2	Génie Industriel	CMP
FOREST	Bernard	PR 1	Sciences & Génie des Matériaux	CIS
FORMISYN	Pascal	PR 1	Sciences & Génie de l'Environnement	SITE
FORTUNIER	Roland	PR 1	Sciences & Génie des Matériaux	SMS
FRACZKIEWICZ	Anna	DR	Sciences & Génie des Matériaux	SMS
GARCIA	Daniel	MR	Génie des Procédés	SPIN
GIRARDOT	Jean-Jacques	MR	Informatique	G2I
GOEURIOT	Dominique	MR	Sciences & Génie des Matériaux	SMS
GRAILLOT	Didier	DR	Sciences & Génie de l'Environnement	SITE
GROSSEAU	Philippe	MR	Génie des Procédés	SPIN
GRUY	Frédéric	MR	Génie des Procédés	SPIN
GUY	Bernard	MR	Sciences de la Terre	SPIN
GUYONNET	René	DR	Génie des Procédés	SPIN
HERRI	Jean-Michel	PR 2	Génie des Procédés	SPIN
INAL	Karim	PR 2	Microélectronique	CMP
KLÖCKER	Helmut	DR	Sciences & Génie des Matériaux	SMS
LAFOREST	Valérie	CR	Sciences & Génie de l'Environnement	SITE
LERICHE	Rodolphe	CR CNRS	Mécanique et Ingénierie	SMS
LI	Jean-Michel	EC (CCI MP)	Microélectronique	CMP
LONDICHE	Henry	MR	Sciences & Génie de l'Environnement	SITE
MALLIARAS	George Grégory	PR 1	Microélectronique	CMP
MOLIMARD	Jérôme	MA	Mécanique et Ingénierie	SMS
MONTHEILLET	Frank	DR 1 CNRS	Sciences & Génie des Matériaux	SMS
PERIER-CAMBY	Laurent	PR 2	Génie des Procédés	SPIN
PIJOLAT	Christophe	PR 1	Génie des Procédés	SPIN
PIJOLAT	Michèle	PR 1	Génie des Procédés	SPIN
PINOLI	Jean-Charles	PR 0	Image, Vision, Signal	CIS
STOLARZ	Jacques	CR	Sciences & Génie des Matériaux	SMS
SZAFNICKI	Konrad	MR	Sciences & Génie de l'Environnement	SITE
THOMAS	Gérard	PR 0	Génie des Procédés	SPIN
TRIA	Assia		Microélectronique	CMP
VALDIVIESO	François	MA	Sciences & Génie des Matériaux	SMS
VAUTRIN	Alain	PR 0	Mécanique & Ingénierie	SMS
VIRICELLE	Jean-Paul	MR	Génie des procédés	SPIN
WOLSKI	Krzysztof	DR	Sciences & Génie des Matériaux	SMS
XIE	Xiaolan	PR 1	Génie industriel	CIS

Glossaire :

PR 0	Professeur classe exceptionnelle
PR 1	Professeur 1ère catégorie
PR 2	Professeur 2ème catégorie
MA(MDC)	Maître assistant
DR	Directeur de recherche
Ing.	Ingénieur
MR(DR2)	Maître de recherche
CR	Chargé de recherche
EC	Enseignant-chercheur
IGM	Ingénieur général des mines

Centres :

SMS	Sciences des Matériaux et des Structures
SPIN	Sciences des Processus Industriels et Naturels
SITE	Sciences Information et Technologies pour l'Environnement
G2I	Génie Industriel et Informatique
CMP	Centre de Microélectronique de Provence
CIS	Centre Ingénierie et Santé

Dernière mise à jour le : 9 mars 2010

Table des matières

Remerciements

La thèse est une école, une aventure, un tremplin aussi.

- Je tiens à remercier particulièrement mon directeur de thèse, Olivier BOISSIER, pour tes conseils toujours pertinents, tes encouragements et ta patience. Ton encadrement a été pour moi certes une école de la recherche mais surtout une école de management.

- Je remercie mon co-encadrant Jomi Fred HÜBNER ainsi que nos collaborateurs italiens Alessandro RICCI et Michele PIUNTI vos contributions ont été déterminantes pour l'aboutissement de ce travail.

- Merci à Gauthier et Yann, ce fut très enrichissant de partager le bureau avec deux encyclopédies ambulantes.

- Merci à l'ensemble de l'équipe ISCOD et du laboratoire LSTI en particulier aux membres de l'ancienne équipe SMA : Laurent, Philippe, Camille, Reda, Yazid pour votre convivialité et vos remarques constructives.

- Un grand merci à l'*École Nationale Supérieure des Mines de Saint-Étienne* en particulier aux services administratifs et à l'école doctorale, vous m'avez permis d'avoir une agréable formation et expérience professionnelle.

- Je remercie mes deux rapporteurs, Marie Pierre GLEIZES et Jaime Simão SICHMAN, dont les remarques et les conseils m'ont été utiles. Vous avoir comme rapporteurs a été un honneur et un plaisir.

- Merci également à Cathérine GARBAY et Flavien BALBO, qui ont accepté d'examiner ce travail. C'était un plaisir de vous avoir parmi les membres de mon jury.

- A mes parents, en particulier ma mère, dont la simplicité et l'humilité qu'elle m'a transmises m'ont permis d'aborder cette thèse avec ingénuité et sérénité.

- Merci à Marie et Brice, il n'y a pas de mots pour vous exprimer ma reconnaissance et mon admiration, je pense que vous le savez.

- Merci à Mme et Mr LARGIER ainsi qu'à toute votre famille pour vos encouragements et votre serviabilité.

- Merci à mes amies stéphanoises : Amelie, Irina, Linda, Marina ainsi qu'à vos conjoints respectifs et à tous ceux que je ne cite pas ; votre votre convivialité et votre soutien m'ont permis de persévérer.

- Merci également à Mr. TCHIAKPE pour tes encouragements.

- Merci aux couples AUBERT, GUYON et ROMEZIN pour votre amitié.

- Merci à chaque membre de ma famille : ma grand mère ; mes tantes et oncles en particulier Mme et Mr FOFIÉ ; mes soeurs et frères : Agnès, Éléonore, Appoline, Guy, Paulin ; mes cousines et cousins ainsi que mes nièces et neveux, vous citer ne servirait qu'à en oublier. Que ce travail contribue à votre enthousiasme dans vos projets.

- Merci à Lyse, Marie-Louise, Nadège, Olga et Raoul ainsi qu'à Chimène, Abakar, Amir, Désiré, ce fut un plaisir de s'encourager mutuellement.

- Merci aux dames de Mont d'Or, aux soeurs du Magnificat ainsi qu'à tous ceux que j'oublie.

- Merci enfin à Celui sans qui je n'aurais peut être jamais pu être ce que je suis et encore moins réaliser ce travail. Puisse Sa grâce continuer à accompagner mon existence.

Chapitre 1

Introduction

L'ouverture s'impose de plus en plus comme une propriété indispensable dans de nombreux systèmes informatiques. Elle est notamment l'une des problématiques des travaux menés dans le domaine des systèmes multi-agents et particulièrement abordée dans notre thèse. Nous allons exposer ci-dessous les différentes motivations nous ayant poussé à aborder cette problématique dans les organisations multi-agents. Ensuite, nous détaillerons les objectifs et la démarche suivie pour nos travaux et nous terminerons ce chapitre par la présentation de la structure du manuscrit.

1.1 Motivations et contexte

Les développements scientifiques et technologiques sont basés sur la conception et le développement de systèmes. La définition de la notion de système dépend du domaine (mathématique, économie, minéralogie, informatique, ...) considéré. Un système peut être simplement défini comme un ensemble d'éléments interagissant entre eux selon un certain nombre de principes ou règles. Les éléments d'un système peuvent être simples ou complexes, avec un fonctionnement centralisé ou distribué. Un système est également caractérisé par une frontière, qui représente les critères déterminant l'appartenance ou non d'un élément au système.

Un système peut être fermé ou ouvert selon qu'il interagit ou non directement avec son environnement. Ainsi, un système est fermé lorsque toutes ses interactions sont entre les éléments appartenant à son environnement ou sa frontière. Dans le cas contraire, c'est-à-dire lorsque certaines interactions sont faites avec des éléments hors de sa frontière le système est dit ouvert.

Dans le domaine informatique, un système est défini comme un ensemble structuré de composants logiciels, matériels et de données, permettant d'automatiser tout ou partie d'un système d'information. Celui-ci offre donc des fonctionnalités à des utilisateurs (personnes et/ou autres composants logiciels). L'ouverture de tels systèmes est généralement définie comme la capacité d'ajouter et d'enlever des fonctionnalités en cours d'exécution [132].

De façon générale, dans des systèmes informatiques fermés, la gestion de l'ajout d'une fonctionnalité ne peut impliquer que des entités internes à l'environnement du système, c'est-à-dire que l'ajout de fonctionnalités appartenant à des entités externes aux frontières du système n'est ni envisagé, ni traité. Quant à la gestion des suppressions de fonctionnalités, dans le cas de systèmes fermés, elle consiste à adapter le système en ne tenant compte que des possibilités de son environnement c'est-à-dire sans aller chercher des solutions auprès d'entités extérieures à ses frontières. En revanche, dans des systèmes ouverts, l'ajout d'une fonctionnalité peut impliquer des entités internes à l'environnement du système, ou des entités externes à son environnement. Pareillement, la gestion des suppressions de fonctionnalités peut être faite par des mécanismes d'adaptation d'entités internes au système, mais aussi elle peut impliquer des entités externes au système. L'ouverture dans les systèmes informatiques implique également la prise en compte des utilisateurs des systèmes par la mise en oeuvre de mécanismes leurs permettant l'accès aux fonctionnalités du système.

Ainsi, l'ouverture s'impose de plus en plus aux systèmes informatiques pour leur permettre d'inter-opérer avec d'autres systèmes, d'être portables vers d'autres systèmes et aussi d'être accessibles par différents utilisateurs. La mise en oeuvre de ces caractéristiques : interopérabilité, portabilité et accessibilité, est généralement facilitée par des conventions et des standards. En effet, comme introduit ci-dessus les interactions dans un système sont régies par des règles. Ces dernières peuvent être différentes d'un système à un autre et développées avec des langages de programmations différents. Ainsi, les conventions et standards définissent des règles communes qui, lorsqu'elles sont prises en compte dans un système, lui permettent de garantir son ouverture.

Dans le cadre spécifique des systèmes multi-agents (SMA) dans lequel s'intègrent les travaux de cette thèse, les principaux éléments du système sont des agents. Ces derniers sont définis comme des entités logicielles situées dans un environnement, et agissant de façon autonome pour atteindre leurs objectifs [86]. L'autonomie d'un agent caractérise sa capacité à agir sans l'intervention d'un tiers (humain ou agent) et de contrôler ses propres actions ainsi que son état interne.

Dans un SMA, les fonctionnalités sont fournies par des agents qui, comme nous

l'avons dit ci-dessus, sont autonomes et ont des capacités de raisonnement et de décision. Ainsi, l'ouverture dans un SMA consiste principalement à gérer les arrivées et les départs des agents n'appartenant pas initialement au système, ce qui implique une problématique d'ajout et de suppression de fonctionnalités fournies par les agents.

En outre, des travaux récents en SMA, proposent une approche de conception et de développement de SMA avec une intégration explicite de l'environnement des agents. En effet, dans les SMA classiques, les agents représentent l'unique objet d'étude. Le fait qu'ils sont *situés* dans un environnement est presque ignoré. Pareillement, il est presque toujours ignoré le fait que l'environnement d'un SMA peut avoir des objets différents des agents et que ces objets fournissent aussi des fonctionnalités. Pour réponse à ce besoin, les travaux d'Alessandro Ricci et al. [117] proposent un modèle de conception de systèmes multi-agents qui s'inspire des environnements humains dans lesquels existent de nombreux objets pouvant réaliser des fonctionnalités indépendantes ou complémentaires à celles des humains. Leur méta-modèle de conception de SMA est basé sur les notions d'*agent* et d'*artefact* (A&A). L'artefact est le concept que les auteurs utilisent pour représenter les objets d'un environnement multi-agent. Chaque artefact d'un SMA encapsule un ensemble de fonctionnalités auxquels les agents peuvent accéder et qu'ils peuvent utiliser dans leur activités. Les artefacts sont des entités passives, c'est-à-dire qu'ils n'ont pas de capacités de raisonnement et de décisions comme les agents. Cette approche a pour avantage de réduire la complexité des agents qui n'ont plus besoin d'implémenter les fonctionnalités qui sont offertes par les artefacts. D'autre part, les artefacts sont conçus de façon à pouvoir offrir un manuel d'utilisation libre d'accès aux agents du SMA auquel ils appartiennent. Ce manuel a pour but de faciliter l'apprentissage de l'utilisation d'un artefact par les agents. Ce qui favorise la mise en oeuvre de l'ouverture dans les SMA.

Dans ce travail, nous nous intéressons plus particulièrement à la manière dont cette problématique est abordée dans les organisations multi-agents. Une organisation multi-agent est définie comme un ensemble d'agent interagissant suivant des schémas de coopération précis pour atteindre des objectifs de l'organisation. Par définition, une organisation a des frontières et donc un environnement qui lui est propre dans lequel évoluent ses agents. Une organisation ouverte est définie comme une organisation dans laquelle des agents autonomes et hétérogènes peuvent entrer et sortir [49, 20]. La gestion de l'ouverture dans une organisation mutli-agent consiste donc à offrir les moyens permettant aux agents d'entrer, d'en sortir tout en tenant compte des objectifs de l'organisation.

L'intérêt de cette problématique aujourd'hui est liée au fait que d'une part, plusieurs travaux [60, 81, 52, 56] ont déjà abordé les problématiques de modélisation et de gestion d'organisations et ont proposé des résultats de différentes approches comme nous le verrons dans l'état de l'art. Cependant, ces travaux ne se sont pas intéressés à la problématique d'ouverture notamment à l'études des besoins et solutions pour la mise en oeuvre de mécanismes de gestion des entrées et des sorties d'agents dans une organisation. En outre, certains résultats proposés peuvent servir de base à la conception de solutions à cette problématique comme nous le verrons dans nos propositions. D'autre part, en considérant l'exigence de plus en plus croissante de nos jours d'ouverture des systèmes et l'application des travaux de recherche en SMA à des cas d'utilisations d'activités de la vie courante comme celui que nous présenterons dans cette thèse, nous réalisons un tout autre intérêt à l'étude et la proposition de solutions pour la mise en oeuvre et la gestion de l'ouverture dans des organisations multi-agents.

Ayant présenté ci-dessus le contexte et les motivations de cette thèse nous allons présenter dans la section suivante les objectifs de nos travaux de recherche.

1.2 Objectifs

L'objectif de nos travaux dans cette thèse est d'établir les besoins relatifs à la problématique d'ouverture dans les organisations multi-agents et de proposer un modèle de spécification et de gestion d'organisations qui assure la mise en place de l'ouverture au sein d'organisations multi-agents.

Notre démarche consistera à analyser les limites des modèles organisationnels existant relativement à l'ouverture en particulier celles des modèles $\mathcal{M}$OISE$^+$ [81] et Moise^{Inst} [68], modèles dont nous nous servirons comme base de conception d'un modèle de spécification d'organisations ouvertes. Ensuite, nous nous inspirerons de l'approches du méta-modèle A&A pour la mise en ouvre et la gestion des propriétés caractéristiques de l'ouverture au sein d'organisations multi-agents. Nous validerons ensuite notre proposition par sa mise en oeuvre avec une application de simulation de la construction d'un édifice à travers laquelle nous présenterons la gestion des entrées des agents qui participent à la construction, la gestion de la coordination de leurs tâches respectives pour la réalisation de l'objectif global de l'organisation ainsi que leur régulation, et enfin la gestion des sorties d'agents dans ce cas d'étude.

1.3 Organisation du mémoire

La suite de ce document est décomposée en trois parties. Dans la première, nous faisons une analyse de l'ouverture d'une part dans les systèmes informatiques et dans les systèmes multi-agents et d'autre part dans le cadre spécifique des organisations multi-agents.

Ainsi, le chapitre 2 propose donc une étude des propriétés de l'ouverture dans des systèmes informatiques en général, ainsi que les particularités correspondantes aux SMA.

Le chapitre 3 présente les caractéristiques et les besoins de l'ouverture dans les organisations multi-agents. En effet, une organisation multi-agent se distingue d'un SMA par le fait qu'une organisation a une structure, des buts que doivent atteindre les agents, et des normes définissant le comportement escompté des agents. Ainsi, gérer l'ouverture dans une organisation multi-agent implique de prendre en compte en plus des besoins liés à la gestion d'un SMA ouvert, ceux spécifiques à la structure, aux buts et aux normes de l'organisation.

Nous terminons cette partie par une synthèse (chapitre 4) dans laquelle nous résumons les caractéristiques d'une organisation multi-agent ouverte. Nous mettrons particulièrement l'accent sur les caractéristiques que nous allons mettre en oeuvre à travers nos propositions dans la partie suivante.

Dans la deuxième partie, nous présentons notre proposition de modèle de spécification et de gestion d'organisations mutli-agents ouvertes. Le chapitre 5 décrit le langage de modélisation d'organisation MOISE qui est une extension de $\mathcal{M}$OISE$^+$ [81].

Ainsi, l'accent sera mis dans ce chapitre sur les nouveaux concepts que nous avons définis afin d'enrichir $\mathcal{M}$OISE$^+$ pour permettre aux organisations spécifiées avec ce langage de mieux gérer leur ouverture.

Le chapitre suivant(6) aborde une des principales caractéristiques de l'ouverture qui est la gestion des entrées/sorties des agents. Nous y présentons une démarche de gestion des entrées/sortie avec la spécification d'une organisation à laquelle nous associons la spécification d'un processus d'entrée/sortie. La spécification d'un processus d'entrée/sortie consistera à définir pour une entité organisationnelle ses portails et portes d'entrée/sortie ainsi que les exigences d'entrée/sortie. Le processus d'entrée ou de sortie est géré par des portails et est supervisé par des agents organisationnels. Un processus d'entrée ou de sortie comprend différentes étapes. Dans le cas des entrées on distingue l'appel à candidature des agents, l'analyse des candidatures enregistrées, la négociation et la validation de contrat pour les agents dont les candidatures ont été

retenues. Dans le cas des sortie, le processus de décompose en demande de sortie, analyse de la demande, négociation et validation de la demande.

Le chapitre 7 présente le modèle de gestion d'organisations ouvertes que nous proposons avec l'infrastructure ORA4MAS. Nous y décrivons successivement les fondements de cette infrastructure, ses principaux concepts qui sont les artefacts organisationnels et les agents organisationnels. Nous présentons également les artefacts organisationnels que nous avons développés pour la gestion d'organisations spécifiées avec le modèle MOISE.

La troisième partie de ce mémoire présente la mise en oeuvre, l'illustration ainsi que les expérimentations de nos propositions. Dans un premier temps (chapitre 8), nous présentons la façon dont les spécifications MOISE ont été développées et encapsulées dans les artefacts organisationnels.

Dans le chapitre 9, nous présentons notre cas d'étude qui est l'application de simulation de construction d'un édifice. Nous montrons les principales étapes de son développement avec quelques scénarios illustrant la mise en oeuvre des propriétés d'ouverture.

Nous terminons ce manuscrit avec une conclusion dans laquelle nous résumons les contributions de cette thèse ainsi que ses limites et ses perspectives.

Première partie

État de l'art

Chapitre 2

Ouverture – Systèmes informatiques et systèmes multi-agents

Dans ce chapitre, nous présentons de façon générale les notions de *système*, d'*environnement* et d'*ouverture*. Nous les spécialisons ensuite au domaine de l'informatique, puis à celui des systèmes multi-agents pour en ressortir les problématiques d'ouverture que nous aborderons dans cette thèse.

2.1 Définitions préliminaires

Système

La notion de système est utilisée dans différentes disciplines, avec à chaque fois une signification spécifique suivant le domaine considéré. C'est la raison pour laquelle il existe une diversité de définitions dans la littérature parmi lesquelles nous pouvons citer :

- *Ensemble cohérent de notions ; Assemblage d'éléments formant un ensemble régi par des lois ; Ensemble de procédés pour produire un résultat* [92].
- *Ensemble d'éléments reliés entre eux* [71].
- *Entité relativement individualisable, qui se détache de son contexte ou de son milieu tout en procédant à des échanges avec son environnement* [134].
- *Ensemble d'éléments interagissant entre eux selon un certain nombre de principes ou règles* [1].

Ces définitions nous permettent, indépendamment du domaine –biologie, économie, physique, social, informatique, etc. –, de distinguer les principales caractéristiques de cette notion de système.

1. Un système est composé d'*éléments*. Ceux ci peuvent être eux mêmes des systèmes devenant alors des sous-systèmes du système considéré [71, 134]. Par exemple, dans le domaine économique, les éléments d'un système sont les capitaux, les produits, les clients, les outils (matériel, humain) de production qui peuvent être constitués de sous-traitants.

2. Les éléments d'un système peuvent être *reliés* entre eux. Leurs liaisons et connexions peuvent produire des résultats. Par exemple, toujours dans le domaine économique, les relations entre les éléments peuvent être catégorisées en : financement, fabrication, vente, service après vente. Les résultats du système sont entre autres, les produits fabriqués, les bénéfices réalisés suite aux ventes.

3. Le système peut avoir des *échanges* avec son *environnement*. Nous présentons quelques définitions de l'environnement d'un système ci-dessous. En considérant l'exemple d'un système économique, son environnement est constitué des vendeurs et des acheteurs, les échanges sont de type achat et vente.

A la suite de cette analyse, il est important de préciser que la nature et le type des éléments d'un système, ainsi que les relations entre les éléments, son environnement et les échanges avec celui-ci, dépendent du domaine du système.

Dans le domaine informatique un *système informatique* se définit comme *un ensemble d'éléments dont les principales fonctionnalités sont de traiter, stocker, acheminer ou présenter de l'information* [1].

Les éléments y sont principalement des ressources de type matériels (PC, serveur, routeur, PDA, ...) ; logiciels (système d'exploitation, SGBD, langage de programmation, ...) et humains (concepteurs, programmeurs, utilisateurs, ...). Les relations entre les éléments sont du type interactions homme – machine, consommation de ressources. L'environnement est constitué d'autres systèmes informatique et donc d'éléments de types matériel, logiciel et humain. La nature des échange entre un système informatique et son environnement est principalement la transmission d'informations à traiter ou traitées.

Ayant ainsi présenté de façon générale ces différentes notions, nous abordons dans le paragraphe suivant, la notion d'environnement introduite brièvement ci-dessus. Celle-ci nous permettra de mettre en exergue les problématiques d'ouverture.

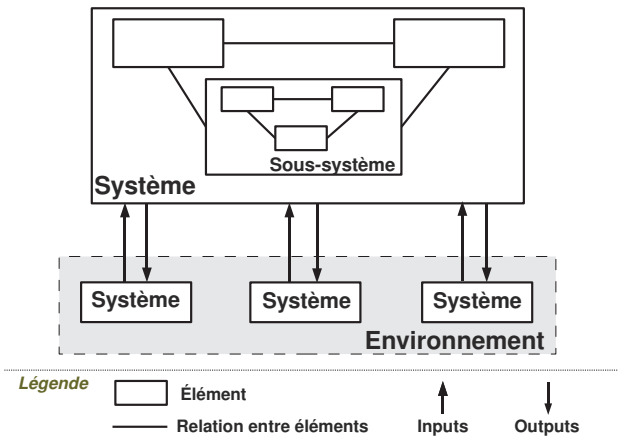


FIGURE 2.1 – Vision d'ensemble de système et environnement

Environnement

Dans le dictionnaire Larousse, un environnement est défini de façon générale comme étant *ce qui entoure* [92] quelque chose ou une personne. Gigch [71] définit l'environnement d'un système comme *tout autre système sur lequel les mécanismes de décision du système n'ont pas de contrôle*. Walliser [134] quant à lui distingue l'environnement interne d'un système de son environnement externe. Ce dernier étant équivalent à la définition de Gigch, tandis que l'environnement interne d'un système est composé de tous ses sous-systèmes ainsi que tout ce qui constitue leurs propres environnements.

Dans la suite, nous adopterons la définition de Gigch et désignerons par environnement d'un système tout ce qui est en dehors de celui-ci et donc n'est pas sous son contrôle. L'environnement sera représenté essentiellement avec les systèmes extérieurs qui ont ou sont susceptibles d'avoir des échanges avec celui-ci (cf. figure 2.1). Les influences réciproques systèmes – environnement sont regroupées par Walliser [134] en deux catégories.

1. Les données échangées :
   - Les *inputs*, qui sont les données introduites dans un système par les éléments de l'environnement.
   - Les *outputs*, données qu'un système génère à destination de son environnement.
2. La modification de la frontière du système :
   - Un système peut étendre sa frontière suite à l'intégration de certains éléments de son environnement.

– La frontière d'un système peut être restreinte par abandon de certains éléments –sous-systèmes– du système.

Exemple, dans le domaine économique, un sous-traitant d'une entreprise peut être racheté et devenir partie intégrante de l'entreprise, et inversement une entreprise peut se séparer d'une de ses unités de production.

En transposant cette analyse dans le domaine informatique, l'environnement d'un système est constitué de toutes les ressources matérielles, logicielles, humaines qui ne sont pas contrôlées par le système. Par exemple, (i) un routeur extérieur à un système, (ii) un utilisateur qui navigue sur un site internet, (iii) une application de recherche telle que Google à laquelle est redirigé un lien de recherche sur une page/site web.

Un système informatique de type réseau peut s'étendre en se fusionnant à un autre. Par ailleurs, une fonctionnalité peut être supprimée dans une application pour être déléguée à une autre application.

En résumé, l'environnement d'un système est lui même composé d'autres systèmes, ceux ci-pouvant échanger des informations et ressources avec le système, ou être intégrés au système et donc influer sur sa structure et/ou sur son fonctionnement. C'est de ces relations entre système et environnement que découlent les problématiques d'ouverture.

Ouverture

L'environnement est central dans la définition de l'ouverture de système. En effet, Gigch définit un système ouvert comme *un système inscrit dans un environnement, interagissant, échangeant et communiquant avec les autres systèmes de cet environnement* [71]. Par contre, Walliser propose la définition suivante, *un système ouvert est un système possédant une entrée et une sortie* [134]. Les notions d'entrée et de sortie ici font référence aux processus du système qui permettent d'une part à son environnement de lui fournir des inputs et d'autre part de générer des outputs destinés à son environnement.

Selon les auteurs, ci-dessus cités, contrairement à un système ouvert, un *système fermé* n'a pas d'environnement, et donc pas d'entrée, ni de sortie. Il n'a pas d'échanges avec des systèmes extérieurs et son état n'est influencé que par ses propriétés et conditions initiales [71, 134].

L'étude de l'ouverture dans les systèmes consiste donc à définir les moyens d'établir et gérer les échanges que peut avoir un système avec les autres systèmes de son environnement. Il s'agit de définir la nature des échanges –échange de biens

(vente, achat), échange de messages (communication)– les données échangées (matériels, informations), les supports et flux d'échanges, les impacts des échanges (stabilité, adaptation, fiabilité). Toutefois, la définition de ces propriétés dépend du domaine considéré mais aussi des fonctionnalités et finalités du système.

Dans le domaine informatique, il n'existe pas de définition unanime de systèmes ouverts. Pour notre analyse nous considérons la définition suivante : *les systèmes ouverts sont des systèmes qui fournissent un ensemble d'avantages en interopérabilité, portabilité, et standards ouverts de logiciels ; ou configurés pour permettre des accès non restreints par des personnes et/ou des ordinateurs* [1].

– L'interopérabilité, définit la capacité d'échange des données et services entre systèmes.
– La portabilité concerne, la capacité et facilité d'intégration d'un système informatique dans un autre. Elle se rapporte à la problématique d'extension d'un système par l'intégration d'autres systèmes de son environnement.
– L'utilisation de standards ouverts permet de faciliter la modification des logiciels utilisés de façon à ce qu'ils puissent mieux répondre aux besoins d'un système.
– Les accès non restreints de personnes et/ou d'ordinateurs dépendent des applications et/ou des systèmes. Par exemple, Internet est accessible par tout ordinateur et toute personne via son ordinateur, mais un système d'information de gestion d'une entreprise ne peut être accessible par toute personne et/ou tout ordinateur.

En faisant une correspondance entre les propriétés générales d'ouverture de systèmes et les propriétés d'ouverture de systèmes informatiques, nous obtenons les similitudes suivantes :

1. Les échanges entre un système et d'autres systèmes de son environnement via des inputs et des outputs se traduisent dans les systèmes informatiques par l'interopérabilités que nous approfondirons dans la section 2.2.1.
2. La possibilité de modification de frontière et d'état d'un système par intégration d'autres systèmes de son environnement ou abandon de certains sous-systèmes correspond dans les systèmes informatiques aux problématiques d'*ajout et suppression* d'entités que nous aborderons dans la section 2.2.2.
3. Par ailleurs, avec la possibilité d'accès non restreints de personnes et/ou d'ordinateurs ainsi que les deux propriétés précédentes nous avons un ensemble

de facteurs qui influencent l'état structurel et fonctionnel des systèmes informatiques ouverts. Ainsi, la stabilité et le bon fonctionnement de ceux-ci nécessitent une régulation et donc des mécanismes de contrôle que nous étudierons dans la section 2.2.3.

2.2 Propriétés d'ouverture de systèmes informatiques

Dans la section précédente, nous avons présenté de façon générale les notions de système et d'environnement qui nous ont permis d'introduire les problématiques liées à l'ouverture des systèmes et en particulier les trois principales propriétés qui résument ces problématiques : interopérabilité, ajout et suppression d'éléments, contrôle. Dans cette section, nous analysons les tenants et les aboutissants de ces propriétés dans les systèmes informatiques.

2.2.1 Interopérabilité

La problématique d'ouverture implique le besoin de faciliter les interactions entre un système et d'autres systèmes de son environnement. Ce besoin se traduit en terme d'interopérabilité, c'est-à-dire permettre à un système d'accéder aux fonctionnalités d'un autre système, d'échanger des informations et de pouvoir utiliser les informations échangées [35, 2]. Il est donc nécessaire d'avoir des structures et interfaces précises de représentation et d'échange de données afin que les interactions et échanges puissent se faire correctement entre systèmes [35]. A cet effet, on distingue dans la littérature trois niveaux d'interopérabilité *syntaxique*, *structurelle*, et *sémantique* [40, 63, 131].

- L'interopérabilité *syntaxique* concerne le format de représentation des connaissances en particulier l'utilisation de format standardisé.
- L'interopérabilité *structurelle* fait référence à l'organisation de l'information au sein d'une ressource qui est régie par son modèle sous-jacent.
- L'interopérabilité *sémantique* est l'interprétation que l'on a d'une représentation d'un domaine fondée sur un consensus.

La prise en compte de ces trois niveaux a entraîné le besoin d'uniformiser la définition de certaines fonctionnalités des systèmes ainsi que leurs interfaces d'interactions afin de garantir l'interopérabilité [1]. Ainsi les consortiums de standardisation et de

normalisation tels que ISO [1], ANSI [2], CEN [3] , AFNOR [4], W3C [5], etc. ont pour objectifs de définir, tester, valider, publier et faire évoluer des normes que peuvent utiliser des constructeurs de systèmes afin que leurs produits –matériels ou logiciels– puissent être interopérables avec ceux d'autres constructeurs.

Une norme est une *règle fixant les conditions de la réalisation d'une opération, de l'exécution d'un objet ou de l'élaboration d'un produit dont on veut unifier l'emploi ou assurer l'interchangeabilité* [92]. Les normes dont il est question ici sont celles établies par un organisme indépendant national ou international qui définit leur caractère universel et limite les modifications unilatérales. Les normes n'imposent pas la façon dont les fonctionnalités d'un système sont conçues et développées dans leur architecture interne. Elles ne définissent que les objectifs des fonctionnalités, leurs données respectives d'entrée et de sortie et les interfaces d'interopérabilités [1]. Par exemple, à travers Internet, les informations transitent d'un réseau à un autre via des routeurs. Ces derniers sont fabriqués par des constructeurs différents. L'utilisation des protocoles standards de transport et transmission des données tels que TCP/IP permet à tout routeur indépendamment de son constructeur et de ses mécanismes internes de fonctionnement de pouvoir envoyer et recevoir des données provenant de tout autre routeur quelque soit son constructeur.

L'interopérabilité est garantie par des normes avec l'avantage qu'il en existe pour pratiquement toutes les étapes du processus d'échange d'informations entre systèmes [1]. Exemples : les normes de codage de caractères telle que UTF-8, ISO-8859-xx qui permettent une interprétation correcte sur des systèmes d'exploitations et les logiciels. Les normes de représentation structurée des données telles que XML, RDFS, etc. Les normes de transport internet et de transmission des données telle que TCP/IP, HTTPS, etc.

Par contre, pour toutes les applications existantes, il n'existe pas de normes qui régissent toutes leurs interactions avec tous les systèmes avec lesquels elles sont susceptibles d'interagir. Ainsi, lorsque des normes ne peuvent être utilisées, les constructeurs et développeurs de systèmes décrivent eux-mêmes et mettent à la disposition de leur environnement les interfaces d'interaction de leurs produits matériels et logiciels ainsi que les représentations des données échangeables.

1. International Organization for Standardization – `http://www.iso.org/`
2. American National Standards Institute – `http://www.ansi.org/`
3. Comité européen de normalisation – `http://www.cen.eu/`
4. Association française de normalisation – `http://www.afnor.org/`
5. World Wide Web Consortium – `http://www.w3.org/`

L'interopérabilité est certes l'une des principales propriétés d'ouverture des systèmes informatiques, mais, elle ne peut suffire à elle seule pour répondre à tous les besoins de cette problématique. En particulier, les besoins relatifs à l'intégration d'un nouvel élément dans un système ou la suppression d'une composante du système.

2.2.2 Ajout et suppression d'éléments

Les propriétés d'ajout et suppression d'éléments sont celles qui influencent fondamentalement un système ouvert [75]. Dans la section 2.1, un système a été défini comme *un ensemble d'éléments reliés entre eux*. Cette définition correspond parfaitement à la nature des systèmes informatiques qui sont généralement constitués de plusieurs entités (sous-systèmes) ayant des fonctionnalités diverses et interagissant entre elles. L'ajout d'une nouvelle entité implique que le système dispose de mécanismes d'entrée et d'intégration. L'intégration implique que les fonctionnalités de la nouvelle entité puissent être explicitement identifiables et que celle-ci (la nouvelle entité) puisse interagir avec les autres entités du système. En effet, les entités d'un système sont généralement structurées et peuvent avoir des fonctionnalités complémentaires. Ainsi, l'arrivée d'une nouvelle entité nécessite une coordination avec celles déjà existantes de façon à ce que le système conserve son bon fonctionnement. De même, la suppression d'une entité dans un système implique que celui-ci dispose de mécanismes de gestion de départ ou de sortie, ainsi que d'adaptation de façon à maintenir une stabilité du système.

Illustrons cette propriété d'ajout et suppression d'élément avec la spécification OSGI [5] mise en oeuvre dans les frameworks Equinox [6] et ApacheFelix [7] tous les deux étant des implémentations open source de OSGI. Ils ont une architecture modulaire de composants appelés bundles. Un *bundle* est une classe Java qui implémente des services ; qui peut être ajouté ou supprimé avant ou pendant l'exécution du système. Le *registry* d'une implémentation OSGI détecte automatiquement l'ajout de nouveaux bundles et donc de leurs services ainsi que la suppression de services suite à la suppression de(s) bundle(s) correspondants. OSGI est donc un exemple de spécification permettant le développement de systèmes ouverts avec des mécanismes d'intégration et d'adaptation dynamique.

La propriété d'ajout et suppression d'éléments pour un système ouvert, favorise d'une part l'extension de ses fonctionnalités [75], d'autre part la contraction de celles-

6. `http://www.eclipse.org/equinox/`
7. `http://felix.apache.org/`

ci vers celles qui lui semblent les plus intéressantes pour sa viabilité. Les mécanismes liés à leur gestion ne sont pas toujours standardisés, ils dépendent généralement des systèmes et de leurs applications finales. Ainsi, de la même façon qu'avec la propriété d'interopérabilité, lorsqu'un système ne dispose pas d'une spécification standard qui décrit les mécanismes d'ajout et suppression d'éléments, il doit lui même fournir explicitement aux systèmes de son environnement, les types et représentations des données ainsi que les interfaces qui permettent l'entrée et l'intégration de nouvelles entités ainsi que celles qui gèrent les suppressions d'entités appartenant au système.

La dynamique liée aux échanges de données avec l'interopérabilité ainsi qu'aux ajouts et suppressions d'entités dans des systèmes informatiques ouverts nécessite des mécanismes qui assurent leur stabilité et robustesse que nous étudions par la propriété ci-dessous.

2.2.3 Contrôle

Le contrôle est défini comme étant *une régulation des opérations d'un système pour que celui-ci réalise les fonctionnalités pour lesquelles il a été conçu* [71].

Dans le domaine informatique, les systèmes et leurs environnements sont constitués de ressources matérielles, logicielles, humaines, et ont pour fonctionnalités de traiter, stocker, acheminer, présenter des informations. Ainsi, les mécanismes de contrôle qui veillent au bon déroulement des traitements et des échanges entre un système et son environnement sont particulièrement importants. Cette importance est justifiée par le fait que l'ouverture expose d'avantages les systèmes informatiques à des influences parfois malveillantes de la part des autres systèmes de son environnement. Ainsi, pour pallier ces influences, les systèmes disposent généralement de mécanismes d'autorégulation c'est-à-dire *des caractéristiques structurelles et fonctionnelles qui sont nécessaires et/ou suffisantes pour qu'un système puisse atteindre ses finalités* [134]. La régulation dans les systèmes informatiques est principalement abordée autour des problématiques de sécurité.

En effet, les informations traitées par un système ou échangées entre systèmes doivent être intègres et fiables et, dans certains cas, elles doivent également être accessibles ou confidentielles en temps opportun [126]. Ainsi, les politiques de sécurité ont pour objectifs d'anticiper et gérer les opérations frauduleuses, non autorisées ou malveillantes lors ou suite à des échanges entre un système et son environnement afin de garantir la fiabilité, l'intégrité, la mise à disposition et la confidentialité des informations de leur système [1].

La mise en oeuvre de la sécurité dans les systèmes informatiques se décompose en plusieurs niveaux qui vont de la gestion des droits d'accès au cryptage des données, en passant par la surveillance du système, les pare-feux, les antivirus et autres moyens [126]. Chacun de ces aspects constitue un domaine de recherche dont nous ne pouvons faire une étude exhaustive dans cette thèse. Le lecteur pourra se référer aux publications suivantes [11, 121, 126, 96].

Par ailleurs, Hewitt [75] préconise d'avoir une gestion décentralisée du contrôle dans les systèmes ouverts c'est-à-dire avec plusieurs entités de contrôle. En effet, avec une seule entité de contrôle, la robustesse du système n'est pas garantie dans la mesure où en cas de surcharge de celui-ci, il peut y avoir rupture de fonctionnement du système, ce qui n'est généralement pas le cas avec plusieurs points de contrôle. En d'autres termes, une répartition des charges de contrôle auprès de plusieurs entités permet de limiter les risques de situations bloquantes de l'ensemble du système. Ainsi, autant le contrôle est important dans un système ouvert, autant il sera efficace avec une gestion décentralisée.

2.3 Ouverture dans les approches RBAC

Après avoir présenté dans la section précédente, les trois principales propriétés qui caractérisent les systèmes informatiques ouverts , nous allons aborder dans cette section une approche généralement utilisée pour préserver un bon fonctionnement des systèmes ouverts appelée RBAC (contrôle d'accès basé sur les rôles). Nous avons choisi d'étudier cette approche car son principe général de gestion de l'ouverture se rapproche de celui utilisé dans les organisations, qui est le champ d'application des travaux de cette thèse.

L'une des politiques de sécurité utilisée dans les systèmes ouverts consiste à définir des droits d'accès pour les différents utilisateurs afin que ceux-ci puissent interagir avec le système dans la stricte limite de leurs autorisations. Les travaux autour du développement de ce type de politique sont connus sous l'abréviation RBAC (Role Based Access Control) [61, 120, 106, 103].

Le principe général d'une politique RBAC [61, 120, 62] peut être résumé comme suit : des *permissions* de réalisation d'*opérations* sur des *objets* sont associées à des *rôles* ; des *utilisateurs* acquièrent des permissions par les rôles qui leur sont attribués. La norme INCITS [8] 359-2004 [62] qui spécifie le standard de politique RBAC définit ses cinq concepts de base de la façon suivante :

8. INCITS : InterNational Committee for Information Technology Standards

- *Utilisateur* : entité –généralement un être humain– qui peut interagir avec le système afin d'y réaliser une opération sur des objets.
- *Objet* : entité qui contient ou reçoit des informations ; exemple : un fichier, toute ressource d'un système.
- *Opération* : programme exécutable dont l'invocation exécute des fonctions ou instructions pour un utilisateur.
- *Rôle* : fonction –généralement professionnelle – qui décrit l'autorité et la responsabilité qu'aura tout utilisateur à qui ce rôle sera attribué dans un système ; exemple : professeur, médecin, étudiant, etc.
- *Permission* : autorisation relative à la réalisation de certaine(s) opération(s) sur un ou plusieurs objet(s) d'un système.

La notion de rôle permet également de hiérarchiser les utilisateurs d'un système et de spécifier des systèmes de type organisation. La norme INCITS 359-2004 [62] précise que plusieurs permissions peuvent être associées à un même rôle et une permission peut être associée à plusieurs rôles ; de même, plusieurs utilisateurs peuvent avoir le même rôle et plusieurs rôles peuvent être attribués à un même utilisateur.

L'une des spécialisations de l'approche RBAC est le modèle OrBAC [3] qui propose un modèle de description de politiques de sécurité basées sur le contexte dans des organisations. OrBac étend le principe général RBAC en définissant trois types supplémentaires de droit d'accès : *obligation*, *interdiction* et *recommandation*. Les politiques de sécurité OrBAC sont représentées par $\alpha(org, r, a, v, c)$: l'organisation org accorde au rôle r le droit α, de réaliser une action a, sur une vue v dans le contexte c [3]. Les valeurs possibles de α sont *permission*, *obligation*, *interdiction* ou *recommandation*. La notion d'opération de l'approche RBAC est remplacée dans OrBAC par *action*, de même la notion d'utilisateur est remplacée par celle de *sujet*. Une *action* est définie dans une *activité* précise. Une *vue* est une utilisation particulière d'un *objet* dans une organisation.

La variété de types de droits d'accès proposée par OrBAC ainsi que les notions de contexte, d'activité, d'action et de de vue permettent de mieux expliciter la modélisation d'une organisation. La figure 2.2 représente l'architecture du modèle OrBAC.

Nous présentons ci-dessous la prise en compte et la mise en oeuvre des propriétés d'ouverture étudiées dans la section 2.2 dans l'approche RBAC.

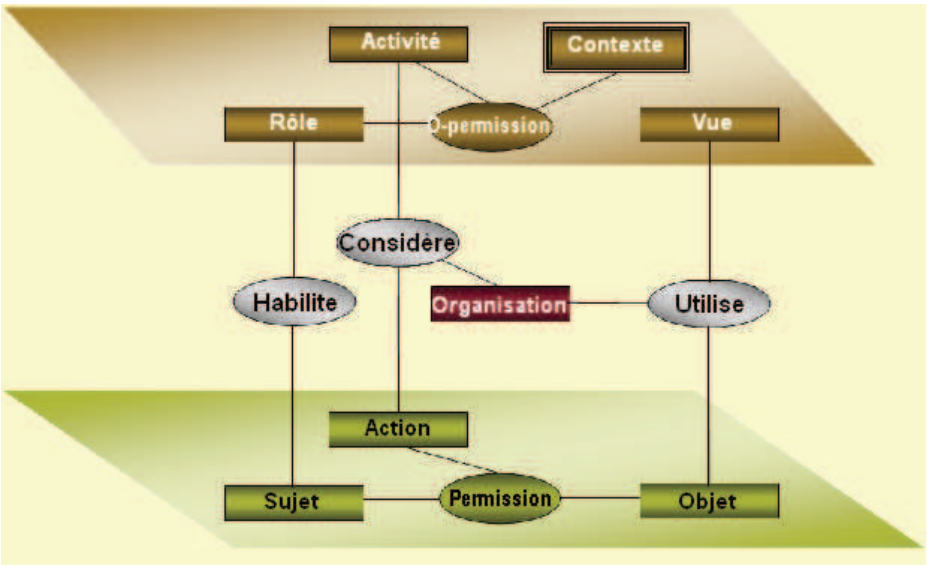


FIGURE 2.2 – Modèle Entité-Relation de l'Architecture OrBAC d'après [3]

RBAC et Interopérabilité

L'interopérabilité d'un système est garantie par des normes. Dans les RBAC, en plus de la norme INCITS 359-2004 [62] qui établit le standard de spécification de politiques RBAC ainsi que leur administration, une proposition de standard d'implémentation et d'interopérabilité (IIS [9] [44, 45] a été soumise à l'INCITS. La proposition RIIS a pour but de définir une norme de conception et implémentation de modèles RBAC conformément au modèle de référence de la norme ANSI INCITS 359-2004. Elle vise également à promouvoir des moyens pour faciliter des échanges de politique RBAC entre modèles différents sans avoir d'ambiguïté au niveau sémantique, de transfert de codes entre implémentations de modèles RBAC et de comparaison de systèmes RBAC.

En attendant que la proposition RIIS soit validée et utilisée, les modèles existant et leur implémentation proposent des interfaces d'interaction qui varient suivant les types de données échangées et selon les choix internes des systèmes. Par exemple, on distingue des interfaces d'authentifications de type lecteur de carte magnétique, et de type boîte de dialogue permettant d'entrer identifiant et mot de passe. L'API MotOrBac [10] du modèle OrBAC, offre sa propre interface de définition et d'administration de politique de droit d'accès,

9. RBAC Implementation and Interoperability Standard
10. MotOrBac : `http://motorbac.sourceforge.net/`

RBAC et entrée – sortie

L'ajout et suppression d'entités dans l'approche RBAC consiste d'une part à gérer les affectations de rôles aux utilisateurs dans un système et d'autre part à gérer les départs des utilisateurs du système. En étudiant la spécification de la norme INCITS 359-2004 [62], nous avons identifié les fonctions ci-dessous qui doivent être implémentées dans tout modèle respectant cette norme. Comme leur nom respectif l'indique, ces fonctions permettent d'enregistrer ou de supprimer un utilisateur : $AddUser(user)$, $DeleteUser(user)$, d'affecter ou de supprimer un rôle à un utilisateur : $AssignUser(user, role)$, $DeassignUser(user, role)$. $USERS, ROLES, UA$ sont respectivement les ensembles des utilisateurs, des rôles du système, ainsi que l'ensemble et des utilisateurs à qui au moins un rôle a été affecté dans le système.

- $AddUser(user)$
  - $user \notin USERS$
- $DeleteUser(user)$
  - $user \in USERS$
- $AssignUser(user, role)$
  - $user \in USERS$
  - $role \in ROLES$
  - $(user, role) \notin UA$
- $DeassignUser(user, role)$
  - $user \in USERS$
  - $role \in ROLES$
  - $(user, role) \in UA$

Cependant, aucun paramètre *même optionnel* permettant de spécifier des conditions d'entrée ou de sortie dans le système et d'affectation ou d'abandon de rôle n'est prévu dans la spécification de ces fonctions par la norme INCITS 359-2004. Ainsi, l'approche RBAC dispose certes d'une prise en compte des propriétés d'ajout et suppression d'utilisateur(s) ainsi que de leur intégration dans un système avec la spécification des primitives dédiées, mais elle ne permet pas de spécifier explicitement des critères d'entrée et de sortie, ainsi que d'affectation de rôles aux utilisateurs, relativement aux objectifs du système. Ces critères nous semblent indispensables en particulier dans des systèmes qui ont des objectifs qu'ils voudraient réaliser –exemple des organisations– et donc qui doivent se prémunir de la réalisation de ces objectifs en vérifiant que les capacités et compétences des utilisateurs qui entrent dans le système et à qui sont affectés des rôles correspondent aux besoins des objectifs du système.

RBAC et contrôle

La norme INCITS 359-2004 [62], prévoit dans sa spécification fonctionnelle des fonctions pour l'administration de politique RBAC, c'est-à-dire de gestion de la dynamique du système ainsi que le contrôle de son état. Parmi ces fonctions, nous pouvons citer :

- *AssignedUsers(role)* : retourne la liste des utilisateurs auxquels *role* a été attribué.
- *UserPermissions (user)* : fournit les permissions d'un utilisateur, aussi bien des rôles qui lui ont été directement affectés que des rôles dont il hérite.
- *CheckAccess(session, operation, object)* : retourne un booléen comme résultat de l'autorisation ou pas d'une opération sur un objet au cour d'une session.
- ...

D'autre part, l'approche RBAC vise à anticiper et interdire toute action non autorisée dans un système par des permissions attribuées aux utilisateurs à travers leurs rôles. Le contrôle du respect des droits d'accès est donc rigoureux voire rigide car pour toute opération si la valeur retournée par la fonction *CheckAccess* est *vrai* alors l'utilisateur pourra réaliser l'opération. Dans le cas contraire, il ne le pourra pas.

Du point de vue méthode de gestion, l'administration et le contrôle de politique RBAC ont évolué dans le temps. On est ainsi passé de méthodes centralisées avec un unique référentiel dans lequel étaient enregistrés et contrôlés tous les droits d'accès d'un système [61] vers des approches de plus en plus décentralisées et récursives c'est-à-dire qui réutilisent l'approche RBAC pour définir et gérer l'administration de politique de droit d'accès. C'est le cas des modèles Admin-OrBAC [46], UAR-BAC [95], ARBAC [119], etc.

En résumé, nous avons présenté dans cette section une approche qui permet la mise en oeuvre et la gestion de l'ouverture dans des systèmes informatiques principalement avec les notions de rôle et de permission. La notion de rôle ici permet de définir des catégories d'utilisateurs à qui est associé un ensemble de permissions. Cette approche a ainsi pour avantage de faciliter la gestion de l'ouverture notamment pour l'entrée, le contrôle et la sortie des utilisateurs. En effet, la gestion des entrées dans un système RBAC revient à gérer les affectations de rôles. Chaque utilisateur ayant un ou plusieurs rôle(s) précis il est aisé de contrôler ses actions ou activités parmi lesquelles la sortie d'un système car celles-ci sont régies par les permissions associées à son (ses) rôle(s). Mais cette approche a quelques limites. Les standards qui permettront de garantir l'interopérabilité sont encore en cours d'étude. La spécification standard de l'approche ne permet pas de définir des exigences pour : l'entrée, l'attribution d'un rôle et la sortie des utilisateurs. Le contrôle est généralement rigide car les implémentations ne permettent en générale pas aux utilisateurs de transgresser les permissions qui leurs sont attribuées par leur(s) rôle(s). Nous étudierons dans la section qui suit ce qu'il en est dans les systèmes multi-agents.

2.4 Ouverture dans les systèmes multi-agents

Les analyses de l'ouverture dans les sections précédentes permettent de présenter notre problématique dans le cadre général des systèmes informatiques. Dans cette section, nous allons orienter notre étude aux systèmes multi-agents qui n'est autre que le domaine spécifique auquel se rapportent nos travaux de recherche.

2.4.1 Systèmes multi-agents

Les systèmes multi-agents (SMA) désignent des systèmes informatiques constitués d'ensemble d'entités logicielles appelés agents qui interagissent, le plus souvent selon des modes de coopération, de concurrence ou de coexistence [9]. Dans cette section, nous n'allons pas faire un état de l'art exhaustif de l'historique et de l'évolution des SMA, les lecteurs peuvent se référer aux publications [138, 86, 127, 9] qui présentent un bonne synthèse des différents résultats de recherche dans le domaine. Cependant, à partir de ces articles, nous nous sommes intéressés aux caractéristiques spécifiques de ces systèmes qui nous permettent d'analyser la problématique d'ouverture. Ainsi, il existe de nombreuses définitions du concept agent plus ou moins équivalentes parmis lesquels nous avons choisi celle de Jennings, Sycara et Wooldridge [138] qui est la suivante.

Un agent est un système informatique, situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu. Les notions : « situé », « autonome » et « flexible » sont définies comme suit :

- situé : l'agent est capable d'agir sur son environnement à partir des entrées perçues qu'il reçoit de ce même environnement.
- autonome : l'agent est capable d'agir sans l'intervention d'un tiers (humain ou agent) et contrôle ses propres actions ainsi que son état interne.
- flexible : l'agent dans ce cas est :
  - capable de répondre à temps : l'agent doit être capable de percevoir son environnement et d'élaborer une réponse dans les temps requis.
  - proactif : l'agent doit exhiber un comportement proactif et opportuniste, tout en étant capable de prendre l'initiative au "bon" moment.
  - social : l'agent doit être capable d'interagir avec les autres agents (logiciels et humains) quand la situation l'exige afin de compléter ses tâches ou aider ces agents à accomplir les leurs.

En plus de ces propriétés, Sycara [127] précise que chaque agent a des capacités

de résolution ou des informations limitées pour la résolution d'un problème. Cette caractéristique est le fondement même des SMA qui est basé sur une décomposition d'un problème complexe en sous problèmes et une distribution de ces derniers à plusieurs agents. Chaque agent résout le sous problème qui lui est confié et l'agrégation des solutions partielles de chaque agent contribue à la solution du problème complexe initial.

Ainsi, les premiers travaux en SMA se sont essentiellement focalisés sur cette problématique de distribution [91, 94, 139]. En effet, un SMA était quasiment un système fermé dans lequel, tous les agents étaient programmés par un même concepteur qui intégrait directement dans leur code toutes les primitives ou fonctions dont ils se serviraient pour la résolution de sous problème(s) dont ils auraient la charge.

Il ne se posait donc pas de problématique d'interopérabilité et plus précisément d'interaction entre agents inconnus [53] c'est-à-dire développés par des programmeurs différents de telle sorte que chaque agent n'a pas de connaissance préalable des agents avec lesquels il interagira dans un SMA.

De même, il ne se posait pas de problématique d'arrivée et départ d'agents inconnus dans un SMA. Tous les agents étaient crées ou étaient auto-crées au lancement du système à partir de prototypes de code qui leurs permettaient de s'exécuter sans mécanismes spécifiques d'intégration.

Quant au contrôle, puisque le concepteur du SMA était également celui qui développait les agents, ces derniers étaient programmés de façon à toujours avoir des comportements bienveillants, c'est-à-dire des comportements qui exécutent les taches ou résolvent les sous problèmes pour lesquels ils ont été programmés sans jamais nuire volontairement à un autre agent du SMA ou au SMA lui même.

Un autre aspect important et souvent négligé dans les premiers travaux en SMA est l'environnement. En effet, l'environnement d'un SMA est constitué d'agents et parfois d'autres ressources que peuvent utiliser les agents [137, 136, 135]. Exemple : un agent robot nettoyeur utilise un capteur pour obtenir les informations lui permettant de déterminer quelles sont les zones à nettoyer, ou une batterie pour se recharger en énergie. Un agent se distingue d'une ressource par le fait qu'il possède des capacités décisionnelles ; tandis que les ressources sont des entités passives et ont des capacités exclusivement fonctionnelles [135]. En d'autres termes, contrairement aux agents, les ressources ne sont pas autonomes. Leurs exécutions se font uniquement par invocation d'instructions ou appels de méthodes par une autre entité –programme, agent–, elles sont programmées pour fournir un certains nombre de fonctions ou services.

Cependant, dans les premiers travaux en SMA, l'environnement était quasiment intégré dans le code des agents. En effet, un agent était programmé avec toutes les

connaissances sur les ressources de son environnement dont il devait avoir besoin au cours de son existence. Cette approche avait pour conséquence de surcharger le code des agents qui aurait pu être évité en envisageant une découverte autonome des ressources nécessaires à la réalisation de tâche de l'agent.

La problématique d'ouverture est donc récente dans le domaine des SMA [49, 20, 8] et comme toute nouvelle problématique, elle a connu des évolutions qui ont abordé progressivement des techniques de gestion d'intégration de nouveaux agents dans un SMA avec le *Broker* [36], des moyen de gestion des interactions entre agents inconnus avec le développement de langage de communication d'agents tel que KQML [11] [90]. Ces travaux ont contribué au développement de standards par le consortium FIPA [12] afin d'uniformiser certains résultats et ainsi de favoriser des recherches vers le développement de SMA de plus en plus ouverts.

Nous approfondirons cette analyse dans les sections qui suivent en étudiant les problèmes d'ouvertures spécifiques aux systèmes multi-agents en suivant le schéma des propriétés présenté dans la section 2.2.

2.4.2 Interopérabilité et hétérogénéité des agents

Afin de voir comment l'interopérabilité est gérée au sein des SMA intéressons nous d'abord à l'hétérogénéité des agents, l'une des propriétés fondamentales des SMA.

Le développement des SMA a été favorisé par de nombreux travaux sur, la conception et l'implémentation des mécanismes permettant aux agents d'acquérir des données, de raisonner, d'agir sur leur environnement ainsi que de communiquer avec d'autres agents. Ces travaux ont abouti au développement de diverses architectures de conception des agents et de différents langages de communication et de programmation d'agents [138].

L'architecture d'un agent définit la façon dont son comportement interne représente les informations qu'il perçoit dans son environnement et prend les décisions qui détermineront les actions qu'il va réaliser. Les architectures les plus connues et utilisées dans la littérature sont celles des agents réactifs et celles des agents BDI (Belief-Desire-Intention) [114]. Les différentes architectures d'agents proposées ont conduit au développement de divers langages et plate-formes de programmation

11. KQML : Knowledge Query and Manipulation Language
12. FIPA : Foundation for Intelligent Physical Agents, www.fipa.org

d'agents chacun correspondant généralement à une architecture particulière. Par ailleurs, l'architecture la plus utilisée est de type BDI et parmi les plate-formes existantes on peut citer Jack [13], AgentTool [14], Jade [15] [15], Jadex [16] [4], Jason [17] [22], 2APL [18] [48].

Cette diversité d'architectures et de langages de programmation d'agents, nous permet de réaliser le potentiel d'hétérogénéité des agents pouvant coexister dans un SMA. Le challenge de l'ouverture dans les SMA relativement à l'interopérabilité est de permettre à des agents ayant des architectures différentes et/ou des langages de programmation différents de pouvoir interagir entre eux et avec les ressources éventuelles de leur environnement.

Ainsi, l'interopérabilité dans les SMA concerne deux types d'interactions : *agent – agent* et *agent – ressource*. Pour ce dernier type, il n'existe pas de normes qui impose la description fonctionnelle des ressources ainsi que leur utilisation. En effet, ce type d'interaction ne constitue pas un engouement particulier de recherches dans le domaine. En général, les programmeurs d'agents implémentent directement dans le code des agents les primitives qu'ils utiliseront pour interagir avec les ressources qu'ils utiliseront. Cette approche pose un problème de découverte dynamique de nouvelles ressources dans un SMA et de capacité à interagir avec elles. Heureusement, des travaux récents [104, 116, 117] proposent une approche intéressante pour la conception et l'implémentation des ressources de SMA avec la notion d'artefact. Le principe est de définir chaque ressource avec la spécification d'un artefact notamment un manuel d'utilisation dans lequel est décrit l'interface d'utilisation de l'artefact. Ainsi les agents, peuvent apprendre eux-mêmes comment interagir avec les artefacts / ressources de leur environnement.

L'interopérabilité pour les interactions agents – agents est abordée avec le développement de langages de communication qui permettent aux agents de pouvoir échanger des messages tout en conservant leur propre architecture. KQML [90, 55] est l'un des premiers langages de communication utilisé par les développeurs de SMA. Plus récemment le consortium FIPA a défini un standard de langage de communication ACL (Agent Communication Language) [64]. Il est basé sur des actes de lan-

13. http://www.agent-software.com/
14. http://macr.cis.ksu.edu/projects/agentTool/agentool.htm
15. http://sourceforge.net/projects/jadepro/
16. http://sourceforge.net/projects/jadex/
17. http://jason.sourceforge.net/
18. http://www.cs.uu.nl/2apl/

gage, exemples : *send, cfp, inform, notunderstood* qui peuvent être utilisés dans des protocoles de communication qui gèrent des échanges de messages entre agents. Ce standard est implémenté dans la plupart des plate-formes de programmation d'agents.

L'avantage à avoir une spécification standard des communications entre agents est de faciliter l'interprétation syntaxique des messages échangés. En effet, grâce aux performatifs utilisés dans les protocoles de communication par les agents au cours de leurs conversations, ils arrivent à se comprendre indépendamment de leur architecture interne et de leur langage de programmation. Par ailleurs, la sémantique des messages échangés dépend des applications. C'est donc aux développeurs d'agents pour une application donnée de préciser quelle est l'ontologie utilisée pour l'interprétation sémantique des messages que les agents échangeront. L'interopérabilité sémantique pour les communications entre agents reste donc une problématique ouverte en SMA malgré les travaux de la DARPA [19] pour la spécification d'ontologie pour les agents notamment le DAML [20] [99] en particulier avec des approches de services web [54, 100].

2.4.3 Entrée–sortie et coopérations des agents

Dans la littérature, un système multi-agent ouvert est généralement défini comme étant un système dans lequel des agents hétérogènes peuvent entrer et sortir facilement [125, 130, 8]. Ainsi, une des préoccupations essentielles dans la conception d'un SMA ouvert est celle de définir les moyens permettant aux agents d'entrer, de s'intégrer et de sortir.

Approche par « Broker »

Un broker est un service de courtage qui assure les fonctions de pages jaunes et de routeur entre les agents d'un SMA. Ainsi, tout agent qui entre dans un SMA s'enregistre auprès d'un broker en précisant ses caractéristiques ou compétences et lorsqu'un agent quitte le SMA, le broker dans lequel il est enregistré le supprime de sa liste d'agents [36]. De plus, lorsqu'un agent a besoin d'un service dont les compétences lui manque, il interagit avec le broker qui lui fournit une liste d'agents qui peuvent réaliser le service sollicité.

Le broker assure les fonctions de routeur lorsqu'il sert d'intermédiaire pour des interactions entre agents [36]. Par exemple, un agent peut envoyer une demande de service au broker qui la transmet aux agents appropriés selon le service sollicité, et inversement pour répondre à une demande de service un agent peut envoyer sa réponse

19. Defense Advanced Research Projects Agency
20. DAML : DARPA Agent Markup Language

au broker qui la transmettra à l'agent destinataire. Notons que la présence d'un broker n'exclut pas les interactions directes agent – agent. En effet, suite à une première interaction entre deux agents avec le broker comme intermédiaire, chacun peut conserver les références(adresse, compétences, ...) de l'autre et ainsi lors de prochaines interactions ne plus faire appel au broker comme intermédiaire.

Par ailleurs, dans des SMA constitués d'un grand nombre d'agents, il est possible d'avoir plusieurs brokers qui gèrent les entrées et les sorties des agents. Cette approche vise à optimiser les recherches d'agents adéquats pour des services spécifiques [23]. Chaque broker ne gère en effet que les entrées et départs d'agents ayant des caractéristiques ou compétences spécifiques.

L'approche de gestion des entrées et des sorties avec broker a un inconvénient qui est la centralisation des données des agents auprès d'un seul composant. Raison pour laquelle des approches de décentralisation des services d'enregistrement et d'intégration ont été proposées parmi lesquelles l'utilisation d'agent(s) accueillant(s).

Approche par « agents accueillants »

La notion d'*agent accueillant* a été proposé par Vercouter [132] en vue d'une gestion collective et coopérative des entrées et intégrations d'agents dans un SMA ouvert. Le principe de cette approche est que tout agent du SMA est un agent accueillant c'est-à-dire :

- Qu'il soit lui même intégré dans le SMA en ayant d'une part enregistré ses compétences et besoins auprès d'un agent accueillant, et d'autre part en étant capable de comprendre les compétences et besoins d'autres agents de façon à pouvoir et leur venir en aide le cas échéant.
- Qu'il ait connaissance des autres agents du SMA de manière à pouvoir d'une part, leur présenter de nouveaux agents –dont il a géré l'entrée– et leur notifier les départs d'agents –dont il a géré la sortie– et d'autre part, qu'il ait un comportement coopératif et bienveillant vis-à-vis des demandes d'entrée et intégration de nouveaux agents.

Ainsi, contrairement à l'approche précédente dans laquelle la gestion d'arrivée et d'intégration de nouveaux agents est gérée par un service dédié (broker) avec le choix d'en avoir un ou plusieurs dans un SMA ouvert, dans cette approche, cette gestion est généralisée ou distribuée auprès de tous les agents du SMA. Bien qu'intéressante, cette approche a en commun avec l'approche broker, le manque de gestion des entrées et sorties de ressources, limite prise en compte dans la recommandation FIPA que nous présentons ci-dessous.

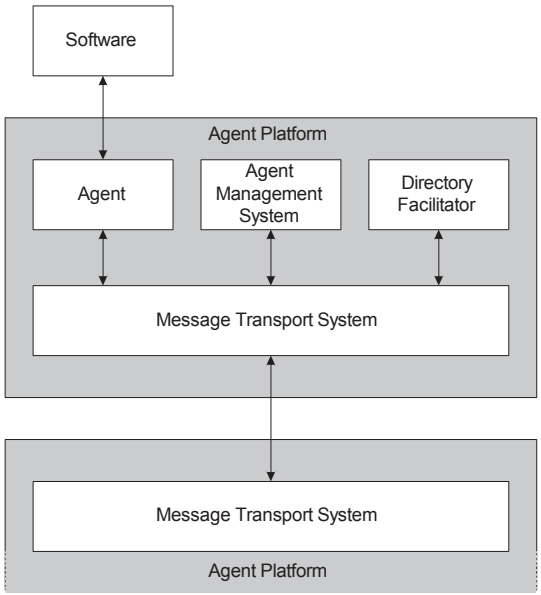


FIGURE 2.3 – Model de référence FIPA de AMS [64]

Recommandations FIPA

Le consortium FIPA [64] de normalisation des domaines agent et multi-agents, propose une spécification de conception de SMA avec une distinction de la gestion des entrées et intégrations des agents de celle des ressources.

Les *entrées et intégrations des agents* sont assurées par un service de pages blanches *agent management service* (AMS) et un service de pages jaunes *directory facilitator* (DF) cf. 2.3. Tout agent qui entre dans un SMA doit s'enregistrer auprès de l'AMS, et enregistrer ses savoir faire –s'il a des services à offrir– auprès du DF. Ce dernier a donc les mêmes fonctionnalités qu'un broker et permet également aux agents de trouver des agents qui peuvent leur offrir des services dont ils ont besoin.

Les *entrées et intégrations des ressources* sont gérées par un *agent wrapper* et un *agent ressource broker* (ARB) cf. 2.3. Un ARB assure un service de pages jaunes pour les fonctionnalités des ressources. Chaque ressource est associée à un agent wrapper, l'ARB publie les fonctionnalités et l'identifiant de l'agent wrapper qui est associé à la ressource qui offre chaque fonctionnalité. Un agent wrapper est un agent qui assure les médiations entre les agents d'un SMA et des ressources non agents. Lorsqu'un agent veut utiliser une fonctionnalité d'une ressource, il envoie un message ACL à l'agent wrapper qui le transcrit en l'instruction correspondante de la ressource sollicitée. Et inversement, l'agent wrapper transcrit les résultats envoyés par une ressource en message ACL qu'il envoie ensuite à l'agent ayant sollicité le service fourni par la

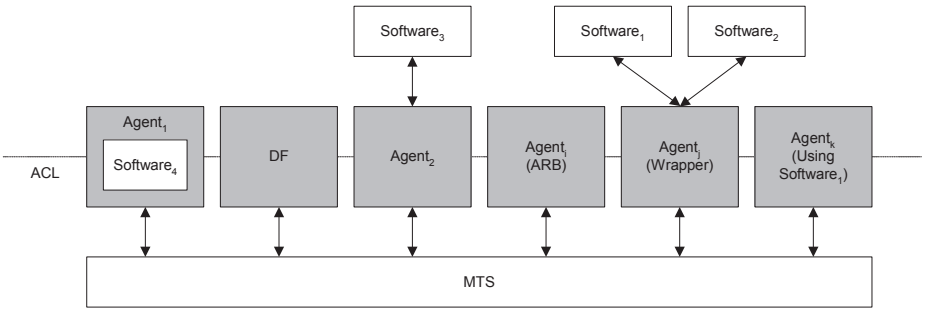


FIGURE 2.4 – Model de référence FIPA des agents wrapper et agent ressource broker [64]

ressource.

2.4.4 Contrôle et autonomie des agents

Les particularités des SMA par rapport aux systèmes informatiques sont : la distribution et l'autonomie des agents. La première permet de répartir la résolution d'un problème complexe au sein de plusieurs agents en vue de la résolution du problème. La seconde consiste à considérer que l'agent peut prendre individuellement ses décisions sans avoir besoin d'une entité de contrôle extérieure (humain ou autre).

Par contre, le comportement des agents n'est pas toujours bienveillant. En d'autres termes, de leur hétérogénéité et autonomie, émergent souvent des comportements non escomptés dans un système. En effet, les agents sont généralement implémentés par des programmeurs différents et chaque agent a ses propres objectifs qui déterminent son comportement.

Par ailleurs, les travaux en SMA ont de plus en plus évolué dans le développement des systèmes qualifiés de sociaux, dans lesquels les coopérations entre agents sont indispensables. La coopération est alors contrôlée par des approches basées sur la confiance et la réputation [140, 101, 59], et des approches de systèmes normatifs [19, 18, 80]. Nous développerons uniquement la seconde approche qui fait partie du cadre de nos travaux de recherche contrairement à la première approche.

L'approche des systèmes normatifs consiste à définir des normes qui régissent le comportement des agents dans une société d'agents. La notion de norme ici et de façon générale dans les SMA, a une sémantique différente de celle que nous avons vue dans la section 2.2 où elle désignait un standard établi par un organisme national ou international de normalisation. En SMA, une *norme* exprime un comportement, plus précisément une obligation, une permission ou une interdiction que des agents

doivent respecter. Elle est définie dans une application particulière et n'a pas de valeur universelle car sa portée se limite en général au cadre de l'application. Cette sémantique de norme en SMA se rapproche de celle de la notion de droit d'accès présentée dans la section 2.3. Dans la suite de ce mémoire, nous utiliserons essentiellement la sémantique multi-agent de la notion de norme.

Le contrôle dans un système multi-agent normatif consiste à vérifier que les agents respectent les normes. Ainsi, on distingue en général deux approches. Une approche préventive qui consiste à n'autoriser que les actions permises à un agent et donc toute tentative d'exécution d'une action interdite par une norme n'aboutit pas [57]. C'est la même approche que celle des modèles RBAC. La deuxième approche est plus flexible car elle autorise la réalisation d'une action interdite et les mécanismes de contrôle analysent alors le non respect de la norme avant de décider d'une application ou non d'une pénalité [76].

Les normes et la structuration dans les SMA constituent les éléments de bases des organisations dont nous étudierons les besoins et moyens de mise en oeuvre de l'ouverture dans le prochain chapitre.

2.5 Synthèse

Dans ce chapitre, nous avons étudié dans un premier temps les concepts de *système* et *environnement* qui nous ont permis de définir la problématique d'ouverture dans un système. Cette étude nous a permis de faire ressortir trois propriétés caractéristiques de l'ouverture dans les systèmes informatiques : l'interopérabilité, l'ajout et suppression d'éléments et le contrôle.

L'étude de ces propriétés nous a permis de révéler que :

- **L'interopérabilité** est la capacité pour un système ouvert d'interagir, échanger des données avec d'autres systèmes.
- **L'ajout et la suppression d'éléments** sont les propriétés qui influencent le plus un système ouvert. Car, la première implique une modification architecturale et fonctionnelle par intégration d'un nouvel élément dans un système, et la deuxième implique une modification identique par suppression d'un élément du système. Ainsi, contrairement à l'interopérabilité qui permet essentiellement l'échange des données à traiter entre systèmes, l'ajout et suppression d'entité ont des finalités d'extension et de restriction architecturale et fonctionnelle dans un système ouvert.

– **Le contrôle** permet à un système ouvert d'assurer sa stabilité et sa robustesse.

Nous avons ensuite étudié l'approche RBAC (Role Based Access Control) relativement à l'ouverture. De cette étude, il résulte que, l'existence d'un standard qui spécifie un modèle de référence de l'approche fait de cette approche une possibilité intéressante de mise en oeuvre de l'ouverture dans des systèmes informatiques. Cependant, nous avons vu que la spécification des fonctions de gestion des entrées et des sorties des utilisateurs a quelques limites liées au fait qu'elle ne permet pas de préciser des conditions d'entrée et de sortie d'utilisateurs ainsi que d'attribution et d'abandon de rôles. En plus, le contrôle avec l'approche RBAC est généralement rigide c'est-à-dire que les utilisateurs ne peuvent en général par enfreindre les permissions qui leurs sont attribuées via leur(s) rôle(s). Quant à la propriété d'interopérabilité, des travaux de standardisation des moyens qui favoriseront les échanges et l'interprétation des politiques RBAC entre des systèmes différents sont encore en cours.

En outre, l'approche RBAC offre l'avantage d'agréger la gestion du contrôle d'accès dans un système avec la notion de rôle puisque plusieurs utilisateurs peuvent avoir le même rôle. De ce fait, les modifications (ajout, suppression) de droits d'accès se font sur les rôles définis dans le système et se répercutent sur les utilisateurs réels à qui ces rôles sont attribués dans le système. Ainsi, il est moins fastidieux de modifier les droits d'accès individuels de chaque utilisateur dans des systèmes ouverts où il est susceptible d'y avoir un grand nombre d'utilisateurs.

Dans les systèmes multi-agents, nous avons vu que les propriétés d'ouverture ci-dessus citées s'articulent avec les propriétés caractéristiques des SMA notamment l'hétérogénéité des agents, leur capacité cognitive et leur autonomie. Ainsi, l'interopérabilité, les entrées et sorties des agents et leur contrôle sont partiellement formalisées et abordées respectivement par des problématiques de langage de communication entre agents, service de pages jaunes et systèmes normatifs.

Les systèmes normatifs, regroupent divers travaux de recherche sur la régulation des agents et en particulier sur la modélisation et la gestion d'organisation multi-agent qui n'est autre que le champ d'application des travaux de cette thèse. Raison pour laquelle, nous allons poursuivre notre étude dans le prochain chapitre avec l'analyse de la prise en compte et de la mise en oeuvre de l'ouverture dans quelques modèles organisationnels multi-agents existants.

Chapitre 3

Ouverture et organisations multi-agents

De nombreux travaux ont été développés et continuent à l'être autour des technologies multi-agents. Ceux-ci portent notamment sur les architectures d'agents, les moyens de communication entre agents, les interactions avec leur environnement, etc. Les technologies ainsi développées constituent des bases pour le développement d'autres problématiques telles que celle des *organisations multi-agents* qui s'intéressent principalement aux mécanismes de coopération entre agents. Dans ce chapitre, nous aborderons dans un premier temps les caractéristiques générales des organisations multi-agents : définition, approches de représentation et de gestion ainsi que les problèmes découlant de leur ouverture. Nous étudierons ensuite quelques modèles organisationnels existant afin d'analyser les moyens qu'ils offrent pour la mise en oeuvre de l'ouverture dans les organisations représentées, déployées et gérées à partir de ces modèles.

3.1 Généralités sur les organisations multi-agents

Dans le chapitre précédent, les agents ont été définis comme étant des entités logicielles autonomes, situées dans un environnement et capables d'avoir des comportements proactifs et sociaux [86]. Ainsi, un agent est capable de décider seul de ses actions, d'interagir avec d'autres agents et avec son environnement. Les premiers travaux de spécification et de mise en oeuvre de ces propriétés d'agents se sont développés dans des contextes de SMA *fermés* dans lesquels, les concepteurs de SMA et les programmeurs d'agents étaient généralement les mêmes [91, 94]. Il ne se posait donc

pas de problème d'échanges de données entre des agents hétérogènes, ni de problème d'intégration d'agents extérieurs au système ou de contrôle de comportements malveillants des agents. La notion d'hétérogénéité des agents est assez large [72, 86, 13] et caractérise un ou plusieurs des aspects suivants : la diversité d'origine c'est-à-dire que les agents sont développés par des programmeurs SMA différents et qu'en début d'exécution, ils ne se connaissent pas d'avance, la diversité d'architectures d'agents, la diversité de langages de programmation utilisés pour développer les agents et la diversité de compétences.

L'inexistence du problème d'hétérogénéité dans les premiers travaux SMA était principalement liée au fait que les applications développées ne nécessitaient par d'interactions entre agents issus de programmeurs différents. L'évolution de l'utilisation des technologies SMA vers ce type des applications plus ouverte a entraîné le développement de travaux sur des langages de communication [90] afin que des agents hétérogènes puissent interagir tout en se comprenant.

Ce premier pas vers l'ouverture a été suivi d'une volonté d'améliorer les recherches d'agents ayant les compétences ou services qui correspondent le mieux aux besoins d'un système ou d'un autre agent. Ainsi, se sont développés des travaux sur les systèmes de courtage [36] et de pages jaunes [64].

La standardisation d'un langage de communication FIPA ACL [64] et d'une architecture de SMA avec un service de pages jaunes par la FIPA [64] a contribué à fournir des bases pour l'ouverture. Ainsi des applications multi-agents peuvent désormais se développer indépendamment de la programmation des agents.

La possibilité d'avoir un service de pages jaunes et un langage de communication commun au sein d'un SMA permet de pouvoir développer indépendamment des agents et des applications. Ces derniers doivent alors se munir de mécanismes de recrutement d'agents dont ils ont besoin dans les pages jaunes et des mécanismes de coordination de leurs activités comme par exemple dans Karma [113]. Cependant, le manque de modélisation permettant d'appréhender la complexité de plus en plus croissante des applications a entraîné la nécessité de méthodologies [141, 123] et de modèles [102] qui favorisent d'avantage le développement de SMA ouverts. De plus, l'hétérogénéité des agents introduit une nouvelle problématique : la malveillance des agents qui jusqu'alors n'était pas considérée. En effet, contrairement aux agents bienveillants qui réalisent les actions / opérations / activités qui leur sont confiées dans une application généralement sans dévier des attentes, les agents malveillants ont des comportements qui ne sont pas souvent ceux escomptés par le système [12, 58, 14, 85].

Les modèles organisationnels apportent donc aux SMA des outils de modélisation plus précisément de structuration des agents, de représentation de leurs répartitions

de tâches ainsi que des règles qui régissent leurs comportements et les régulations qui en découlent [74]. La modélisation permet ainsi de réduire la complexité des problèmes traités ainsi que de faciliter les modifications et les adaptations des aspects logiciels [74].

- La *structuration* est généralement définie dans les modèles organisationnels – dont nous étudierons certains plus loin dans ce chapitre– avec des concepts tels que *équipe, groupe, scène* et *rôle*. De façon générale, des agents jouent un ou plusieurs rôle(s) dans un(e) ou plusieurs équipe(s), scène(s) ou groupe(s).
- La *répartition des tâches* découle d'une décomposition des objectifs de l'organisation en sous-objectifs et tâches qui sont ensuite associés aux rôles. Ces tâches et sous-objectifs seront réalisés par les agents auxquels auront été attribués les rôles correspondants.
- Les règles de *régulation* sont définies par des *normes* qui établissent les permissions, obligations, interdictions des rôles et donc des agents au sein d'une organisation.

Définition : organisation multi-agent

Malgré les nombreux travaux qui abordent les problématiques de modélisation et de gestion d'organisations, il n'existe pas de définition commune de la notion d'*organisation*. Parmi les définitions existantes dans la littérature, nous pouvons citer les suivantes :

- Une organisation est un schéma de décision et de communication appliqué à un ensemble d'acteurs qui accomplissent un ensemble de tâches pour satisfaire des buts tout en gardant un état cohérent du système [97].
- Une organisation est caractérisée par une décomposition de tâches, une distribution de rôles, un système d'autorité, un système de communication et un système de récompense et de pénalisation [17].
- Une organisation est un système structuré avec des schémas d'activité, connaissance, culture, mémoire, histoire, et capacité qui sont distincts de ceux d'un seul agent [67].

Apparemment différentes, les diverses définitions de la notion d'organisation présentent cependant des similitudes car, on retrouve généralement dans chaque définition les caractéristiques de structuration, de répartition des tâches et de régulation avec des formulations différentes.

Ainsi, dans le cadre de cette thèse, nous définissons une organisation multi-agent de la façon suivante.

Définition : une organisation mutli-agent est un ensemble d'agents qui interagissent suivant une structure, des schémas de coopération et des normes prédéfinis pour atteindre des objectifs précis.

Cette définition illustre le fait que dans nos travaux, nous nous intéressons essentiellement aux approches de développement de modèles organisationnels dites *centrées organisations* [20]. En effet, dans les SMA deux grandes communautés abordent la problématique des organisations multi-agents. La communauté SASO [1] (*Self-Adaptive ans Self-Organising systems*) [109] dont les approches sont dites *centrées agents* [20] car elles s'intéressent principalement au développement de comportements locaux à chaque agent et d'interactions pair-à-pair adéquats dont le résultat émergeant constitue une organisation [124, 16, 93, 108]. Le processus de construction de telles organisations est ascendant ou bottom-up. Dans la communauté COIN [2] (Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems) les approches de développement d'organisation sont de type descendantes car elles sont centrées sur les organisations. En effet, les organisations sont préalablement spécifiées ainsi que les mécanismes de coordination des comportements des agents. Concrètement, contrairement aux approches utilisées dans la communauté SASO, où le fonctionnement des organisations peut conduire à *des phénomènes imprévisibles et a priori non vérifiables formellement*, les approches utilisées dans la communauté COIN visent à assurer *certains invariants comportementaux des agents* [107, 108].

Après cette mise au point de la distinction des deux principales approches de développement d'organisations multi-agents, nous pouvons constater que notre définition d'une organisation présentée ci-dessus introduit les deux aspects qu'abordent les approches descendantes. D'une part, on a besoin de modéliser de façon explicite la *structure*, les *objectifs* et leurs décomposition en tâches ainsi que leur répartition aux agents. D'autre part, on a besoin de gérer les organisations concrètes tout au long de leur cycle de vie, c'est-à-dire de leur *création* à la *réalisation* des objectifs en passant par les *évolutions* structurelles et fonctionnelles. La considération de ces deux aspects a conduit à une distinction des deux composants d'un modèle organisationnel multi-agent appelés respectivement *Organisation Modelling Language* (OML) et *Organisation Management Infrastructure* (OMI) [20].

Ainsi, de nombreux travaux proposent des modèles organisationnels qui offrent soit l'un des deux composants, soit les deux avec des concepts et des approches variées dont nous étudierons quelques uns dans la section 3.4. Afin de permettre une meilleure

1. http ://www.saso-conference.org
2. http ://www.pcs.usp.br/ coin/

compréhension de cette étude, nous présentons d'abord ci dessous les notions de OML et OMI.

3.2 Langage de modélisation d'organisation

Un langage de modélisation d'organisation (OML) est un langage s'appuyant sur un modèle organisationnel et permettant de représenter la structure d'une organisation, les schémas de coopération des agents, et éventuellement les normes qui régissent les répartitions rôles – tâches et les comportements escomptés des agents [20].

La représentation de chacun de ces éléments constitue une *dimension* d'une organisation [81, 42]. La notion de dimension a été introduite par le modèle Moise+ [81] dont l'OML représente une organisation en trois dimensions : structurelle, fonctionnelle et déontique permettant de décrire respectivement la structure, les schémas de coopération et les normes d'une organisation. La notion de dimension a ensuite été utilisée pour généraliser les décompositions proposées par des OML dans la littérature pour représenter des organisations. L. Coutinho et al dans [42] ont ainsi dénombré cinq principales dimensions proposées par différents modèles [3] : structurelle, fonctionnelle, contextuelle, dialogique et normative.

Dimension structurelle

Elle permet de décrire la décomposition d'une organisation en équipes [128, 112] ou groupes [60, 81, 52, 47] ou scènes [56, 66] dans lesquels sont définis des rôles que des agents pourront adopter.

Dimension fonctionnelle

La définition des objectifs d'une organisation ainsi que leur découpage en sous-objectifs et tâches avec l'ordonnancement qui établit leurs dépendances et l'ordre de réalisation des tâches est faite dans la dimension fonctionnelle [56, 81, 128, 52, 47]. Ainsi, c'est à travers cette dimension que les agents raisonnent sur leur potentiel de coopération et de participation à la réalisation des objectifs d'une organisation.

Dimension contextuelle

Cette dimension permet de décrire les différentes situations dans lesquelles les agents pourront se retrouver au cours de leur évolution dans l'organisation [128, 56,

3. Dans [43], un nombre supplémentaire de dimensions a été proposé. Dans ce document nous nous restreignons à celles identifiées initialement dans [42].

OML	Structurelle	Fonctionnelle	Contextuelle	Dialogique	Normative
AGRS [118]	++	–	–	+	–
BRAIN [25]	+	–	–	–	–
EV [32]	+	++	–	–	++
Islander [56]	+	–	++	+++	++
Moise+ [81]	+++	+++	–	–	++
Moise^{Inst} [68]	+++	+++	++	–	+++
OperA [52]	++	++	+	++	+++
RIO [123]	+	+	–	+++	–
ROAD [39]	+	–	–	+	–
Teamcore [112]	++	++	–	–	–
THOMAS [6]	++	++	–	–	++

TABLE 3.1 – Quelques OML avec leurs dimensions. Les signes –, +, ++, +++ signifient respectivement que : la dimension n'existe pas dans l'OML ; La dimension existe dans l'OML avec une faible expressivité ; La dimension existe dans l'OML avec une expressivité moyenne ; La dimension existe dans l'OML avec une forte expressivité. (EV = entreprise virtuelle). Tableau inspiré de [42]

68]. Elle est donc constituée de contextes qui servent à décrire ces différentes situations ainsi que les comportements des rôles que pourront avoir les agents qui les ont adoptés [129, 68].

Dimension dialogique

Les communications entre les agents au sein d'une organisation peuvent être décrites par un ensemble de protocoles et accompagnées d'une ontologie qui précise la sémantique des messages échangés [56, 52, 47]. La description des protocoles et de l'ontologie représentent la dimension dialogique d'un OML.

Dimension normative

Les comportements des agents sont contraints par des normes qui définissent leurs droits ou permissions, leurs devoirs ou obligations et leurs interdictions au sein d'une organisation. L'ensemble de ces normes forme la dimension normative [56, 52, 68].

Le tableau 3.1 présente une synthèse des dimensions proposées dans quelques OML de la littérature. Chaque dimension peut avoir une expressivité plus ou moins riche selon l'OML considéré.

La description d'une organisation avec un OML constitue une *spécification organisationnelle* (OS [4]) [81]. Malgré la divergence d'expressivité des OML, chacun représente un langage de description d'organisation. Á partir des OS, des agents hétérogènes peuvent raisonner sur des organisations et établir leurs stratégies de participation à la réalisation des objectifs de celles-ci. Ainsi, les OML contribuent à considérer les organisations comme un moyen d'ouverture des SMA.

Après avoir présenté ce qu'est un OML et en quoi il constitue un facteur pour l'ouverture, nous présentons dans la section suivante, le deuxième élément d'un modèle organisationnel à savoir l'OMI qui est l'infrastructure de gestion des activités des agents dans des organisations.

3.3 Infrastructure de gestion d'organisation

L'infrastructure de gestion d'organisation (OMI) est une plate-forme qui permet de créer et de gérer des entités organisationnelles dont l'OS est décrite avec l'OML pour laquelle elle a été implémentée [20]. Une *entité organisationnelle* (OE [5]) est une instance d'organisation constituée d'une spécification organisationnelle (OS), d'agents et de tout ce qui contribue à la gestion de l'état de l'instance de l'organisation [20, 68]. Nous avons présenté dans la section 3.2 une OS comme étant la description de la structure et des objectifs d'une organisation, ainsi que des règles qui régissent l'évolution de son état et qui contraignent l'autonomie des agents.

Afin de permettre la réalisation des objectifs d'une organisation, les OMI gèrent divers services organisationnels qui assurent le déroulement des coopérations entre agents au sein de l'organisation. Ces services concernent l'attribution des rôles aux agents que se soit dans des équipes comme dans Teamcore [112], dans des groupes avec Madkit [73] et S-MOISE+ [83], et dans des scènes dans AMELI [57] ; La gestion des interactions des agents avec l'organisation, des coordinations des activités des agents et donc leurs interactions / communications avec d'autres agents et enfin la régulation des agents qui inclut des récompenses et sanctions des comportements des agents selon les normes définies dans l'organisation [89].

L'étude de quelques OMI [88, 21] a révélé que leur architecture de gestion des services ci-dessus cités correspond généralement à l'architecture générale présentée

4. OS : Organisation Specification
5. OE : Organisation Entity

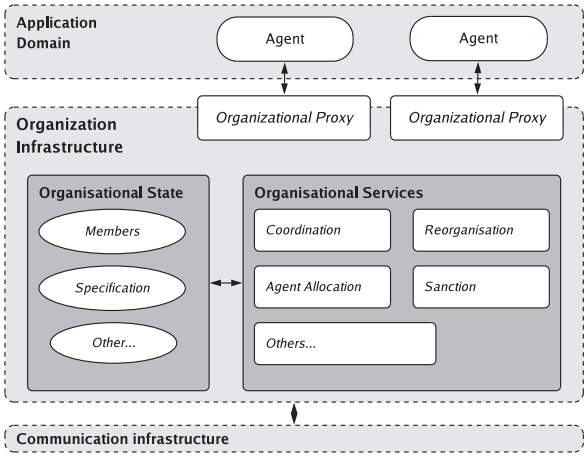


FIGURE 3.1 – Architecture générale des OMI [89]

par la figure 3.3. On y distingue une couche de *proxies organisationnels* qui s'intercalent entre les agents d'applications et les services organisationnels qui gèrent les entités organisationnelles. Chaque OMI propose son propre type de proxy : *Governor* pour AMELI [57], *OrgBox* pour S-Moise+ [83], *TeamcoreProxy* pour Teamcore [112] *AgentInRole* pour Madkit [118]. Tous ces proxies ont le même objectif, celui de faciliter les interactions entre les agents d'applications généralement hétérogènes et des organisations gérées par l'OMI. En effet, l'ensemble des primitives ou protocoles de communication dont les agents ont besoin pour leurs interactions avec l'organisation sont fournis par les proxies qui assurent donc les médiations entre les agents d'applications et les services organisationnels en transmettant les actions de l'agent aux services organisationnels concernés par ces actions et en restituant à l'agent les réponses de l'organisation transmises par les services organisationnels [89].

Quant aux services organisationnels de coordination et de régulation –sanctions, récompenses–, ils veillent au bon déroulement des coopérations entre les agents d'applications et à la réalisation des objectifs des entités organisationnelles.

En résumé, avec les OML et les OMI les modèles organisationnels offrent des moyens de développer des applications multi-agents beaucoup plus ouvertes que ce ne fut le cas avec les premiers travaux en SMA. Cependant, la question qu'on est en droit de se poser est celle de savoir si les OML et OMI répondent aux propriétés d'ouverture que nous avons présentées dans le chapitre 2 à savoir l'interopérabilité, les entrées / sorties et de contrôle au sein des organisations ? Si oui comment ? et sinon

quelles sont leurs limites face à ces propriétés ? Nous essaierons de répondre à ces questions dans les sections ci-dessous.

3.4 Démarche d'analyse de l'ouverture dans les organisations SMA

R. Scott [122] considère qu'une organisation est un système ouvert selon la définition suivante : « Une organisation est une coalition de groupes d'échanges d'intérêts qui réalisent des buts par négociation ; la structure de la coalition, ses activités et ses résultats sont fortement sous l'influence des facteurs de l'environnement ».

L'influence de l'environnement sur une organisation évoquée dans cette définition rejoint celle de notre analyse de l'ouverture des systèmes présentée dans le chapitre précédent. Dans ce même chapitre, l'environnement d'un système a été défini comme étant tout ce qui est à l'extérieur du système et n'est pas sous le contrôle du système. De la même façon, l'*environnement d'une organisation multi-agent* comprend tous les agents et ressources qui n'appartiennent pas à l'organisation et donc qui ne sont pas sous le contrôle de celle-ci.

Ainsi, nous étudierons dans cette section comment les éléments spécifiques aux organisations SMA notamment la structure, les objectifs, les schémas de coopération et les normes (cf. *définition organisation* présenté section 3.1) décrits par un OML et gérés par une OMI, sont influencés par les propriétés d'ouverture, et comment ces éléments influencent l'environnement d'une organisation. Nous allons étudier ces influences relativement aux propriétés d'interopérabilité, d'entrée et de sortie des agents ainsi que de contrôle au sein d'une organisation multi-agent.

Interopérabilité

L'interopérabilité d'un système (cf. section 2.2.1) caractérise ses interactions avec des entités de son environnement c'est-à-dire, les échanges de données que le système peut avoir avec ces entités et la capacité d'utilisation de ces données [1].

Dans le cas d'organisations multi-agents, les interactions entre des agents extérieurs et une organisation concernent essentiellement des demandes d'OS, des demandes de procédures d'entrée / sortie et des demandes d'informations relatives à l'état de l'OE (membres de l'organisation, objectifs en cours de réalisation, manager, etc.). En effet, les agents extérieurs à une organisation ont besoin de savoir comment elle est structurée afin de savoir par exemple quel rôle ils peuvent adopter. Lorsqu'il existe une décomposition structurelle en groupes, scènes ou équipes, l'OS permet éga-

lement aux agents extérieurs de savoir à quelle entité structurelle (équipe, groupe, ...) ils peuvent être associés au sein d'une organisation. D'autre part, avec l'OS les agents identifient les objectifs et services d'une organisation et peuvent ainsi mieux orienter leurs choix de coopérations avant d'entrer dans une organisation [30].

Ainsi, l'étude de l'interopérabilité syntaxique de modèles organisationnels revient à analyser si les modèles tiennent compte du besoin d'accessibilité plus ou moins total à l'OS et l'OE d'une organisation par des agents extérieurs ainsi que l'accessibilité à la procédure explicite d'entrée / sortie.

Les échanges entre une organisation et des agents de son environnement se font à travers des interfaces d'échanges. Les modèles organisationnels doivent donc permettre aux agents extérieurs de demander ou accéder à des données qui leur permettront éventuellement d'entrer dans une organisation. Inversement, des organisations doivent avoir des outils qui gèrent ces échanges avec leur environnement. L'interopérabilité structurelle de modèles organisationnels concerne donc principalement les OMI. Ces derniers doivent disposer de composants ou mécanismes qui représentent les interfaces d'échanges entre une organisation et son environnement.

Quant à la troisième caractéristique de l'interopérabilité à savoir la capacité d'utilisation des données échangées, elle est fonction d'une ontologie commune entre agents et organisation de la sémantique des données. Son étude dans un modèle organisationnel consiste à déterminer si ce modèle utilise une ontologie standard ou s'il fournit les protocoles d'échanges de données avec leur sémantique et le domaine d'interprétation des données échangées afin de garantir une compréhension des échanges entre des agents extérieurs et une organisation.

Entrées / sorties d'agents

Une organisation multi-agent est caractérisée par le fait qu'elle a des objectifs qu'elle voudrait réaliser et qu'un seul agent ne peut réaliser. Ainsi, elle a donc tout intérêt à admettre l'entrée d'agents qui pourront lui permettre d'atteindre ses objectifs. De ce fait, l'entrée des agents dans une organisation n'est pas aussi simple que dans une plateforme SMA comme nous l'avons présenté dans le chapitre précédent. Concrètement, il ne suffira pas à des agents de s'enregistrer auprès d'un service de pages jaunes d'un SMA pour être considérés comme membres d'une organisation. En effet, une organisation étant structurée que ce soit uniquement avec des rôles ou avec des groupes, équipes ou scènes dans lesquels sont définis des rôles, lorsqu'un agent entre dans une organisation, il doit être intégré dans sa structure. En plus, les agents

qui entrent dans une organisation doivent participer à la réalisation des objectifs de cette organisation.

Cependant des modèles organisationnels, permettent de décrire et gérer des organisations avec des natures d'objectifs différents. Par exemple, des organisations de type e-commerce ont l'objectif de permettre que les transactions d'échanges de biens entre agents se déroulent correctement [10]. Des organisations de type gestion d'une conférence scientifique où les objectifs sont de permettre aux auteurs de soumettre des articles à la conférence et de permettre à cette dernière de gérer entre autre le processus de sélection des articles [49], etc. Dans chaque type d'organisation on a donc une procédure d'entrée et d'intégration différente en fonction des objectifs et de la représentation structurelle –rôle uniquement ou entité structurelle et rôles– de l'organisation.

En ce qui concerne la sortie, les procédures varient également d'un modèle à un autre bien que les problèmes qui en résultent soient presque les mêmes dans les organisations. En effet, l'ouverture impose de permettre à des agents déjà membres d'une organisation de pouvoir en sortir. Cependant, puisqu'un agent est intégré dans la structure d'une organisation et qu'il participe à la réalisation des objectifs de celle-ci, la gestion d'une sortie nécessite de considérer les impacts que celà peut avoir sur la structure et les objectifs de l'organisation afin de remédier lorsque c'est nécessaire et possible aux situations d'instabilité et de dysfonctionnement. A cet effet, nous étudierons également quelques propositions qui sont faites par des modèles organisationnels dans la section suivante.

Contrôle au sein d'une organisation

Le contrôle est parfois considéré comme la principale caractéristique des organisations en général et donc des organisations SMA. En effet, une organisation permet de structurer un système et de préciser les responsabilités, droits et devoirs des membres de ce système. Ainsi, les organisations multi-agents sont souvent assimilées à des systèmes normatifs dont les particularités sont d'établir des normes, de veiller à leurs respect et à leur éventuelles évolutions. Par contre, tous les modèles organisationnels multi-agents ne sont pas normatifs (cf. tableau 3.1). Les modèles qui n'ont pas la dimension normative font ainsi une hypothèse forte sur la bienveillance des agents. Cette hypothèse est d'autant plus forte que les agents sont autonomes. Ainsi on distingue avec ce type de modèle d'une part, une approche de contrôle qui contraint quasi totalement l'autonomie des agents c'est-à-dire qu'aucune action imprévue dans le système ne peut être réalisée par un agent. D'autre part, une approche de gestion d'organisation sans aucun mécanisme de contrôle. Les organisations sont alors su-

jettes aux bienveillances ou malveillances des agents ce qui constitue un grand risque avec l'ouverture.

Les enjeux du contrôle pour les modèles organisationnels multi-agents sont donc de pouvoir représenter ce qui est à contrôler notamment les actions, coopérations, interactions des agents vis-à-vis d'une organisation ; d'établir des règles qui régissent et permettent de gérer les contrôles et de mettre en oeuvre des techniques appropriées aux besoins des organisations.

Après avoir présenté les particularités correspondant aux propriétés d'ouvertures au sein des organisations multi-agents, nous allons maintenant étudier un peu plus en détail les propriétés d'entrée / sortie et de contrôle à partir de modèles concrets avec leur OML et OMI. Par souci de réduire la longueur du texte correspondant à l'analyse de chaque modèle, nous avons choisi de ne pas présenter l'étude de l'interopérabilité modèle par modèle. La synthèse de l'étude de cette propriété pour les différents modèles est présentée dans la section 3.7 et le tableau 3.2.

Les modèles que nous présentons dans les sections 3.5 et 3.6 sont ceux dont les approches de gestions des propriétés d'ouverture nous semblent pour certains originales, montrant ainsi la diversité des mécanismes de mise en oeuvre de ces propriétés et, pour d'autres pertinentes par rapport à une réelle prise en compte des aspects de l'ouverture.

3.5 Ouverture et organisations non normatives

L'intérêt de plus en plus croissant des applications pour des organisations multi-agents a entraîné de nombreuses propositions de modèles organisationnels ces dernières années. Nous présentons dans cette section quelques modèles non normatifs et dans la section 3.6, nous étudierons des modèles normatifs. En effet, nous remarquerons que la présence ou pas de la dimension normative dans un modèle influence aussi bien les procédures d'entrée / sortie que celles de contrôle.

3.5.1 RIO

L'OML *Rôle Interaction Organisation (RIO)* [98, 123] propose une approche de spécification de protocoles d'interactions entre agents dans des organisations ouvertes. Un *protocole d'interaction* est globalement décrit par des *micro-rôles* intervenant dans le protocole et pour chaque micro-rôle les *compétences* –ensemble de messages entrant et sortant– associées. Les micro-rôles sont regroupés en sous ensembles de *rôles*

composites. Ces derniers sont à leur tour agrégés pour constituer des *agents abstraits* qui sont ensuite affectés à des organisations. La notion d'*agent abstrait* utilisée dans se modèle ressemble à celle de *rôle* utilisée dans d'autres modèles. Il n'y a pas de structuration d'organisations en équipes ou groupes ou autres. En résumé, une organisation décrite avec le modèle RIO gère des interactions entre agents.

L'infrastructure de gestion d'organisation (OMI) du modèle s'appelle G qui est une extension de la plateforme MAGIQUE [123].

La procédure de gestion des entrées est très simple. Avec l'OMI G, les agents sont créés et enregistrés directement dans la plateforme à partir d'un prototype d'*agent élémentaire « vide »* . Ils s'enrichissent ensuite en acquérant dynamiquement de nouvelles compétences fournies par la plateforme. Une compétence est implémentée comme un service avec les technologies OSGI. Une compétence étant un ensemble de messages entrants et sortants que peut traiter un micro-rôle, les compétences sont acquises selon les types d'agents abstraits –puisqu'un agent abstrait est composé de rôles-composites, qui sont eux-mêmes composés de micro-rôles. Cependant, les compétences ne précisent pas d'exigences pour l'entrée dans une organisation. La procédure d'entrée a une gestion centralisée car tous les agents sont crées par l'unique *PlateformAgent* de l'OMI. Seul le nom de l'agent est fourni en input à la primitive de création et enregistrement d'un nouvel agent. A l'issue d'un processus d'entrée, l'agent est considéré comme un membre de l'organisation (*OrgMember*).

Quant à la procédure de sortie, les auteurs du modèle n'en font aucune mention. Par contre, ils proposent qu'un *agent doit pouvoir quitter temporairement le système et retrouver sa place ultérieurement*. L'idée est alors de conserver les messages entrants de l'agent afin qu'il puisse les traiter lorsqu'il reviendra.

Concernant le contrôle des comportements des agents, l'OML RIO ne permet pas de spécifier de normes. Et pourtant, on pourrait imaginer qu'en raison de leur autonomie, certains agents peuvent ne pas toujours se comporter selon la spécification des compétences de l'agent abstrait qu'il représente. En effet, un agent peut décider de ne pas répondre à un message entrant ou d'y répondre avec un message non prédéfini dans l'une de ses compétences. Dans ces cas le modèle semble déléguer le contrôle aux agents puisqu'il ne prévoit pas de moyen de contrôle géré par le *PlateformAgent*.

Discussion

Le principal intérêt du modèle RIO est que des agents hétérogènes peuvent entrer facilement dans une organisation, s'y intégrer en acquérant dynamiquement des compétences. Cependant, cette approche semble supposer que tous les agents qui entrent dans une organisation sont bienveillants puisque le modèle n'a pas de mécanisme de

contrôle.

3.5.2 ROAD

Après avoir vu un premier modèle avec une gestion des entrées des agents très simple intéressons nous à celle proposée par ROAD [39, 38].

Les organisations spécifiées avec l'OML ROAD sont constituées d'un ensemble de rôles. Les buts de chaque rôle sont explicitement décrits. Pour certains rôles les procédures de réalisation de ses buts sont également décrites. Le choix de fournir ou non à un rôle des procédures de réalisation de ses buts est lié au degré d'autonomie qu'une organisation veut accorder aux agents à qui seront attribués ce rôle. En effet, la particularité du modèle ROAD est qu'un degré d'autonomie des agents peut être associé à un rôle. Ainsi, pour certains rôles, les agents n'auront aucune autonomie, ils devront alors réaliser les buts de ce type de rôles selon les procédures définies –parfois implémentés– par l'organisation. Pour d'autres rôles l'organisation laissera aux agents l'autonomie de réaliser les buts de ces rôles selon les procédures de leurs choix.

La procédure d'entrée dans une organisation spécifiée avec ROAD est basée sur l'adoption d'un rôle. Pour cela, tout agent qui effectue une demande d'entrée doit fournir les exigences du rôle sollicité. Une *exigence* est définie par les auteurs du modèle comme étant les connaissances et capacités de l'agent à pouvoir réaliser les buts d'un rôle en fonction de l'autonomie associée à ce rôle. Par contre, les auteurs ne précisent par comment doivent être représentées des exigences ni quelles sont les procédures de vérification correspondantes. A l'issue de la procédure d'entrée, l'agent devient un OrgMember s'il est admis dans l'organisation car, pareil qu'avec RIO, il n'y a pas de possibilité explicite de structuration d'organisations en groupes avec ROAD. Un contrat est établi pour l'agent dans lequel sont enregistrés les parties signataires du contrat (les identifiants respectifs de l'agent et de l'organisation), les protocoles d'interactions entre l'agent et l'organisation ainsi que les clauses relatives aux engagements de chacun des signataires du contrat [38].

Aucune mention n'est faite par les auteurs sur la gestion des sorties. Par contre, ils ont à peu près la même approche de gestion de vacance d'un rôle que celle du modèle RIO. La différence avec l'approche de ce dernier est que contrairement à RIO dans ROAD, les messages sont associés aux rôles et donc en cas de vacance d'un rôle (parce que l'agent qui le jouait l'a quitté temporairement ou définitivement), les messages à destination du rôle sont sauvegardés et seront traités lorsque le rôle sera à nouveau attribué.

Pour le contrôle des agents au sein d'une organisation, l'approche est basée sur le

degré d'autonomie accordé aux agents. Par exemple, avec les rôles pour lesquels aucune autonomie n'est accordée aux agents qui les adoptent, le contrôle se fait implicitement de façon ad-hoc car ces agents ne pourront pas réaliser d'actions non autorisées à leurs rôles. En effet, pour tout rôle dont l'organisation décide de n'accorder aucune autonomie aux agents qui le joueront, un composant implémentant les procédures de réalisation des buts du rôle est fourni aux agents qui se contenteront de les exécuter sans possibilité de les modifier. Dans le cas des rôles où une autonomie partielle ou totale est accordée aux agents, l'approche de contrôle est basée sur les clauses de contrat et par des agents dédiés appelés *supervisor*.

Discussion

L'originalité de ROAD est l'intégration de l'autonomie dans la procédure d'entrée dans une organisation. Cet aspect nous semble être un élément supplémentaire à l'ouverture car un agent peut choisir d'entrer ou pas dans une organisation suivant l'autonomie qu'il souhaite avoir pour la réalisation des buts des rôles qu'il adoptera au sein de l'organisation. La gestion des vacances de rôles est également un point positif car elle contribue à la robustesse des organisations ouvertes.

3.5.3 BRAIN

Le modèle *Behavioural Roles for Agent INteractions* (BRAIN) [25] propose une spécification et gestion des interactions entre agents à travers le concept de rôle. Pour chaque rôle est défini un ensemble de capacités et de comportements que devront avoir les agents qui joueront le rôle. Les notions de (a) capacité et de (b) comportement sont respectivement définies dans ce modèle comme (a) un ensemble d'actions que peut faire un agent pour réaliser ses objectifs ; (b) un ensemble d'événements que l'agent qui joue le rôle est supposé pouvoir gérer afin de se comporter en accord avec la spécification du rôle.

Il existe deux OMI pour ce modèle RoleSystem [26] et RoleX [24]. Pour les deux, l'approche de gestion des entrées est basée sur l'*adoption d'un rôle*. La spécification des capacités et des comportements des rôles ne constitue pas des exigences à remplir par les agents puisque, les capacités et comportements des rôles sont implémentés et fournis aux agents. Les différences entre les deux implémentations RoleSystem et RoleX se situent au niveau de la procédure de gestion des entrées / sorties.

Dans la première implémentation RoleSystem [26], le composant chargé de cette gestion est appelé *ServerAgent*. La procédure se résume à la création d'un certificat correspondant au rôle sollicité par l'agent appelé RoleRegistration. Ce certificat est en

fait une implémentation des actions et événements du rôle. L'agent s'en servira donc pour réaliser toute action relative au rôle et pour percevoir les événements issus de ses interactions. Ainsi, les outputs de la procédure d'entrée sont : le statut de l'agent qui devient OrgMember et son certificat.

La gestion de la sortie est aussi simple que celle de l'entrée. En effet, lorsqu'un agent n'a plus besoin d'un rôle, il le signale au ServerAgent qui annule la validité du certificat correspondant à ce rôle.

Dans la deuxième implémentation RoleX [24] (Role eXtension for agents), la gestion des procédures d'entrée / sortie est faite par le composant RoleLoader. La procédure d'entrée est atypique puisqu'une opération de fusion des codes binaires de l'agent et du rôle est réalisée. A l'issue de celle-ci, l'agent peut donc faire appel aux actions et services du rôle comme si l'implémentation du rôle était la sienne.

Quant à la gestion des sorties, elle n'est pas évoquée. Seule la procédure d'abandon d'un rôle est sommairement décrite. En effet, un agent informe le RoleLoader de son intention d'abandonner un rôle. Le RoleLoader effectue alors l'opération inverse de fusion pour séparer le code de l'agent de celui du rôle qu'il veut abandonner.

Dans chacune des implémentations la gestion des entrées / sorties est centralisée par le ServerAgent pour RoleSystem [26] et par le RoleLoader pour RoleX [24]. Cependant, leur OML Brain ne prévoit pas de spécification de norme pour le contrôle des agents. Les approches d'implémentation des OMI confirment ce choix d'absence de contrôle puisque les agents n'ont pratiquement pas d'autonomie avec dans RoleX la fusion des codes d'agent et de rôle et dans RoleSystem l'utilisation par l'agent d'un *certificat* qui implémente les fonctionnalités d'un rôle.

Discussion

Le modèle de spécification d'organisation proposé par BRAIN a un souci de description de façon explicite les capacités et comportements des rôles. Ce souci permet de faciliter aux agents hétérogènes les choix des rôles qu'ils voudraient adopter en entrant dans une organisation. Mais, comme dans le modèle RIO, les fonctionnalités des rôles sont implémentées et fournies aux agents. S'il est vrai que cela facilite l'intégration des agents après leur entrée dans une organisation, cette approche a pour inconvénient de limiter presque totalement l'autonomie des agents. Car ces derniers ne peuvent pas utiliser leur propre implémentation des capacités et fonctionnalités des rôles qu'ils adoptent. En conséquence, des mécanismes de contrôle ne sont pas prévus par le modèle.

3.5.4 AGRS

L'OML AGRS (Agent Groupe Rôle Service) [118] est une extension du modèle AGR [60]. Contrairement aux modèles présentés précédemment, AGR et AGRS permettent de spécifier une organisation avec une dimension structurelle constituée de *groupes* et pour chacun d'eux l'ensemble de leurs *rôles*. Chaque rôle représente une (ou un ensemble de) fonction(s) de son groupe. Dans AGRS, pour chaque rôle on définit ses *services*. *Un service est la description des fonctionnalités que doit proposer l'agent qui se déclare pouvoir offrir le service ou que cherche l'agent qui souhaite l'obtenir* [118].

Dans le modèle AGR, la gestion des entrées est basée sur l'adoption d'un rôle. Six cas de procédure d'admission dans une organisation sont distingués dans ce modèle.

1. *L'admission ou le refus automatique* dont la procédure d'admission est généralement basée sur des conditions facilement calculables ou vérifiables. Par exemple l'admission d'agents pour le rôle spectateur d'un match dont la principale raison de refus d'admission est le nombre maximum de billets déjà vendus, soit la cardinalité maximale du rôle spectateur.
2. *L'admission contrainte par l'état courant du groupe*. Ici l'état du groupe peut être comme dans le cas précédent le nombre de membres, ou les similarités entre les agents candidats et les agents déjà membres du groupe.
3. *L'admission conditionnée par les compétences*, l'idée ici est d'associer un ensemble de compétences nécessaires à un rôle. Tout agent qui veut jouer ce rôle devra donc avoir ces compétences.
4. *L'admission contrainte par l'architecture*, l'architecture ici fait référence à celle des agents. Ainsi un agent qui veut entrer dans l'organisation doit avoir une structure interne correspondant à celle exigée dans l'organisation.
5. *L'admission soumise à un dialogue préliminaire*, la démarche d'entrée ici est basée sur une négociation entre un agent qui veut entrer dans l'organisation et un agent gestionnaire des entrées.
6. *L'admission soumise à une position sociale préalable* le principe est de permettre à un agent d'adopter un rôle que s'il est déjà titulaire d'un autre rôle dans l'organisation.

La distinction de ces six cas montre une diversité des motifs d'admission dans une organisation et donc des mécanismes pouvant être mis en oeuvre pour leur gestion. Le choix d'une procédure dépend de l'application considérée. Il est possible d'avoir plusieurs cas dans une même application suivant ses types de groupes et de rôles.

Cependant, les auteurs de ce modèle ne fournissent pas de spécification permettant de représenter et de gérer les exigences, les inputs, les outputs des procédures correspondantes à chacun de ces cas d'admission. La gestion des sorties d'une organisation n'est pas évoquée dans ce modèle.

Dans le modèle AGRS, la gestion des entrées est basée sur les concepts de groupe et de rôle. En effet, un agent peut entrer dans un groupe soit en adoptant un rôle du groupe, soit lorsqu'elles existent, en remplissant les conditions d'entrées dans le groupe. Ainsi, il est possible de définir des conditions d'entrée dans un groupe notées *Cds*, et des conditions de jouer le rôle notées *Cdj*. Par contre, les auteurs du modèle ne donnent pas de précisions sur la représentation des conditions Cds et Cdj, ni sur les procédures de vérification de ces conditions. Des services étant associés à des rôles, un agent qui sollicite jouer un rôle doit fournir ses capacités à pouvoir offrir les services du rôle.

Dans le cas d'entrées dans des groupes basées essentiellement sur des Cds, l'analyse d'une demande d'entrée consiste à vérifier que l'agent qui effectue la demande remplit les conditions Cds. Lorsqu'il s'agit d'entrer par un rôle, l'analyse des demandes vérifie que la cardinalité maximale du rôle n'est pas atteinte ainsi que les Cdj du rôle concerné par la demande sont satisfaites. Les entités responsables de la gestion des entrées sont le composant *knowledges pumps* pour le groupe secondé éventuellement par un agent manager. Pour chaque rôle, un *RoleManager* gère les adoptions de ce rôle. Lorsque la procédure d'entrée dans un groupe se passe bien, l'agent devient membre du groupe (*groupMember*) et lorsqu'il adopte un rôle un proxy appelé *AgentInRole* est crée et lui est affecté. L'agent effectuera alors toute action relative au rôle à travers son proxy. Un agent a autant d'AgentInRole que de rôle distinct qu'il joue dans une organisation.

Le modèle AGRS comme son modèle parent AGR n'a pas de dimension normative. Ainsi, le contrôle des comportements des agents semble être géré de façon adhoc.

Discussion

L'extension proposée par AGRS pour le modèle AGR a pour avantage de permettre une spécification explicite des types de services d'un rôle notamment ses services *offerts, fournis et transférables*. Mais le modèle ne propose pas de moyen de spécification explicite des conditions Cdj, et des Cdu. De plus, le fait qu'un agent ait autant de proxy *AgentInRole* que de rôle qu'il joue dans une organisation nous semble être un inconvénient majeur pour l'ouverture lorsqu'on suppose qu'on considère la propriété

de robustesse. En effet, dans une organisation de grande taille, c'est-à-dire avec un grand nombre de rôles et d'agents, le nombre d'AgentInRole géré par l'OMI pourrait être exponentiel ce qui pourrait entraîner une surcharge des traitements et une rupture de fonctionnement de l'organisation.

3.5.5 TeamCore

L'OML Teamcore [112] est basé sur le principe de travail en équipe. Une organisation décrite avec ce modèle a une dimension structurelle constituée d'équipes et de rôles. Pour chaque équipe l'ensemble de ses rôles est organisé de façon hiérarchique. Les objectifs d'une organisation sont repartis en arbres hiérarchiques de tâches appelés plans. Les agents participent à la réalisation des tâches d'une organisation en jouant des rôles dans des équipes.

La procédure d'entrée avec ce modèle est basée sur l'attribution d'un rôle dans une équipe. L'approche de gestion est différente de celle utilisée dans la majorité des modèles où la démarche d'entrée est faite par les agents qui veulent entrer dans une organisation. Avec Teamcore, c'est l'organisation qui prend l'initiative de rechercher dans les pages jaunes du SMA les agents dont les expertises ou compétences correspondent aux besoins de ses rôles. Par contre, les auteurs du modèle ne précisent pas comment doivent être représentées les expertises ou compétences nécessaires pour chaque rôle, ni comment la procédure d'analyse des compétences des agents par rapport à ceux des rôles est fait. L'entité de l'OMI chargée du recrutement des agents est le composant Karma.

Les agents ayant été admis sont des OrgMembers de l'organisation et des TeamMember dans leurs équipes respectives. Un *Teamcore proxy* est attribué à chaque agent pour la médiation de ses actions dans l'organisation. La gestion de la procédure de sortie n'est pas évoquée dans ce modèle.

Quant au contrôle, le modèle ne prévoit pas de spécification de normes. Cependant, l'OMI veille à la bonne réalisation des tâches attribuées à chaque agent à travers un rôle. En effet, lorsqu'un agent abandonne une tâche, les composants de l'OMI cherchent un agent remplaçant et lui attribue la tâche.

Discussion

Comme pour les autres modèles présentés précédemment, le modèle Teamcore a une volonté d'expliciter les tâches associées à un rôle. Cependant les exigences correspondant à l'attribution d'un rôle ne sont pas exprimées de façon explicite. Pourtant cela permettrait probablement d'avoir une double possibilité de gestion des entrées

au sein d'une organisation et de ce fait une plus grande ouverture des organisations. Car en plus d'avoir la démarche d'aller chercher les agents qui ont les compétences dont elle a besoin, la procédure d'entrée pourrait également se faire à l'initiative des agents qui dans ce cas proposeraient leurs compétences à l'organisation en fonction de la spécification des exigences des rôles. Le manque de norme est également une limite quand à la spécification explicite des règles de contrôle.

3.6 Ouverture organisations normatives

Les modèles normatifs offrent la possibilité de spécifier de façon explicite des normes qui contraignent les comportements des agents au sein d'une organisation. Selon les modèles, les normes caractérisent également plus ou moins les approches de gestion des entrées / sorties et de contrôles.

3.6.1 THOMAS

Le modèle THOMAS [7], propose une approche de développement de SMA ouverts basée sur les propriétés et technologies orientées services. Une organisation spécifiée avec cet OML a une décomposition structurelle en *unités organisationnelles* (OU). Chaque OU est composée d'un ensemble de rôles auxquels sont associés des services par des normes. Les services sont des entités logicielles offrant des fonctionnalités décrites et implémentées avec des technologies orientées services. Les agents entrent dans une organisation en adoptant un rôle afin d'offrir les services correspondants à ces rôles. La correspondance entre rôles et services est définie par des normes.

La gestion des entrées est faite dans l'OMI par le composant OMS (*Organisation management services*) qui gère de façon centralisée toutes les entrées dont la procédure est basée sur l'adoption d'un rôle. Un agent qui voudrait entrer dans une organisation fournit en input uniquement son identifiant et celui du rôle qu'il sollicite. Après attribution du rôle à l'agent, un contrat est établi entre l'organisation et l'agent pour le rôle adopté. Cependant, les auteurs ne précisent pas comment est représenté un contrat ni les données qui y sont enregistrées. L'agent devient membre de l'unité organisationnelle (UnitMember) dans laquelle il a adopté le rôle et OrgMember de l'organisation.

Quant à la procédure de sortie, elle n'est pas explicitement définie, seul la procédure d'abandon d'un rôle est décrite. Ainsi, lorsqu'un agent veut quitter un rôle, l'OMS vérifie qu'aucune norme interdisant l'abandon du rôle n'est active. Si cette condition est vérifiée, le rôle est supprimé de la liste des rôles de l'agent.

Le contrôle est basé sur la spécification des normes faites dans une organisation. Une norme est définie comme suit :

$$\begin{aligned} \langle norm \rangle &::= \langle deontic \rangle \langle entity \rangle \langle action \rangle \left[\langle temporal \rangle \right] \left[IF \langle if\_condition \rangle \right] \\ \langle ext\_norme \rangle &::= \left[SANCTION \left(\langle norm \rangle \right) \right] \left[SANCTION \left(\langle norm \rangle \right) \right] \end{aligned}$$

Les valeurs possible de $\langle deontic \rangle$ sont : ***obligged, forbidden, permitted***. L'entité ($\langle entity \rangle$) à laquelle la norme s'applique qui peut être soit un *rôle*, soit une OU. Les données $[\langle temporal \rangle]$ et $[IF \langle if\_condition \rangle]$ sont optionnelles et permettent de définir respectivement une condition temporelle et contextuelle d'activation d'une norme. Une norme est éventuellement accompagnée des spécifications des normes qui décrivent la sanction et la récompense qui lui sont associées. Les normes sont enregistrées auprès de l'OMS à travers le service $RegisterNorm$ dont la spécification est la suivante :

$$RegisterNorm(NormID, AdressedRole, Content, IssuerRole, \\ [DefenderRole], [PromoterRole])$$

$AdressedRole$ est le rôle des agents qui fourniront le service. $Content$ est la spécification du service à fournir. $IssuerRole$ est le rôle d'un agent chargé de vérifier la réalisation ou pas du service. $DefenderRole$ est une donnée optionnelle qui précise le rôle de l'agent chargé de sanctionner en cas de non réalisation du service, $PromoterRole$ est également une donnée optionnelle qui précise le rôle de l'agent chargé de récompenser en cas de réalisation du service.
Cette spécification montre une approche qui offre la possibilité de distribuer le contrôle au sein d'une organisation avec la possibilité d'exprimer de façon explicite des rôles distincts pour les agents qui constatent le respect et le non respect des normes, ainsi que ceux qui sanctionnent et ceux qui récompensent.

Discussion

Ce modèle normatif, propose un OML qui permet d'une part de structurer une organisation en rôles et ensemble de rôles appelé *unité organisationnelle (OU)*, et d'autre part de définir des services ainsi que des normes qui associent rôles et services. Cependant, il n'existe pas de moyen de représenter les exigences permettant aux agents de prouver qu'il ont la capacité d'offrir les services d'un rôle qu'ils voudraient adopter. Ce choix est lié au fait que les services sont implémentés et mis à la disposition des agents comme des compétences dans la modèle RIO ou les *services fournis* du modèle AGRS. Ainsi, cette approche a le même avantage que celui que nous avons déjà évoqué dans les modèles ayant à peu près la même approche à savoir qu'elle facilite

l'intégration des agents après leur entrée dans une organisation. Elle a également le même inconvénient c'est-à-dire qu'elle ne favorise pas l'autonomie des agents puisque ceux-ci ne peuvent pas offrir ou utiliser leurs propre services mais uniquement ceux fournis pas l'OMI. Cependant la dimension normative du modèle THOMAS permet d'avoir un contrôle moins rigoureux des comportements des agents, avec la possibilité de sanctionner ou récompenser.

3.6.2 ISLANDER

Le modèle Islander [56] permet de décrire et représenter des interactions dans des institutions électroniques. *Une institution électronique (EI) est une organisation d'acteurs autonomes au sein de laquelle leur comportement est influencé par des normes supervisées par un système d'Arbitrage* [68]. Le modèle d'EI proposé dans Islander a une décomposition structurelle en scènes définies par le *performative structure*. Chaque scène comprend des rôles que peuvent adopter des agents. Le modèle propose de pouvoir définir des conditions d'accès / adoption d'un rôle (WA_r) et des conditions d'abandon de rôle (WE_r) mais aucune spécification explicite n'est fournie pour la représentation et la gestion de ces conditions. Les scènes sont reliées entre elles avec des conditions de transitions. Les *normes* d'une EI définissent pour chaque rôle les obligations d'élocutions à utiliser dans une scène. L'ensemble des élocution valides dans une EI est défini dans un *dialogic framework*.

Les entrées / sorties dans l'EI sont gérées par l'institution manager (IM) et dans une scène par un scène manager (SM) et un transition manager (TM). Un proxy appelé Governor est attribué à chaque agent à son entrée dans une EI et l'agent devient un institution member (InstMember). Un Governor gère toutes les interactions et actions de son agent dans une EI, que ce soit des demandes d'entrée dans une scène ou les envois de message de coopération entre agents. La procédure d'entrée dans une scène est basée sur l'adoption d'un rôle. Puisqu'une spécification organisationnelle d'EI avec l'OML Islander est un workflow d'interactions au sein d'une scène et de transition entre scènes, l'entrée dans une EI est matérialisée par une entrée dans la scène initiale de l'EI dans laquelle il n'y a pas de rôles. C'est de cette scène initiale que les agents pourront accéder aux autres scènes de l'EI en ayant rempli à chaque fois les conditions de transition d'une scène à une autre. Les conditions de transition sont vérifiées par le transition manger (TM) associé à la transition. Le SM quant à lui vérifie que la cardinalité maximale du rôle sollicité n'est pas atteinte. Lorsqu'un agent entre dans une scène, il est membre de celle-ci (ScMember).

En ce qui concerne la procédure de sortie, lorsqu'un agent veut quitter une scène

il informe son Governor qui vérifie si aucune norme n'interdit la sortie de l'agent de la scène. Si c'est le cas, le Governor communique avec un Transition Manager (TM) pour la gestion de la sortie de scène. Par contre lorsqu'un agent veut sortir d'une EI, c'est l'IM qui gère la sortie à cet effet, il vérifie que l'agent n'est plus ScMember d'aucune scène.

Les normes qui régissent le contrôle dans Islander sont définies de la façon suivante : $Obl(r, \psi, s)$, signifient que le rôle r est obligé de réaliser l'action ψ, dans la scène s. Cette spécification a la particularité de permettre de préciser le contexte –la scène– dans lequel la norme est en vigueur. D'autre part, l'approche de contrôle des comportements des agents relativement aux normes d'une EI est totalement préventive. C'est-à-dire qu'un agent est obligé de respecter les normes qui lui sont associées.

Discussion

L'approche de l'OML Islander ressemble à celle de RIO. Avec RIO, on spécifie des *protocoles d'interactions* entre micro-roles tandis qu'avec Islander, on spécifie des *élocutions* entre rôles mais les élocutions sont contextualisées dans des scènes. La gestion des entrées n'est pas explicitement définie mais l'intégration des agents hétérogènes est prise en compte avec les proxies « *Governor* ». Cependant les gestions du contrôle et des sorties au sein d'une EI est assez rigide.

3.6.3 OPERA

Le modèle OPERA [51] permet de spécifier des organisations structurées en groupes et en scènes. La notion de *groupe* désigne simplement un ensemble de rôles. Une *scène* est définie avec un *ensemble de rôles*, les *états* que devront atteindre les agents qui joueront ses rôles, les *schémas d'interactions* qui doivent permettre d'atteindre ces différents états et les normes qui régissent les comportements des agents dans la scène. Pour chaque rôle sont définis l'ensemble de ses *objectifs* ou buts, les plans ou *sous-objectifs* du rôle ainsi que les *droits* et les *normes* du rôle. Un rôle a également un type dont les valeurs possibles sont *externe* ou *institutionnel*. Un rôle externe peut être adopté par des agents qui ne sont pas déjà membres de l'organisation, tandis qu'un rôle institutionnel ne peut être adopté que par les agents déjà membres de l'organisation. L'OML Opera a également une dimension dialogique qui permet la spécification de l'ontologie et des actes de communication d'une organisation.

La procédure d'entrée dans une organisation est basée comme dans le modèle précédent sur l'adoption d'un rôle. Tout agent, qui voudrait adopter un rôle doit satisfaire les exigences de celui-ci en fournissant ses buts et ses plans qui seront comparés à

ceux du rôle qu'il veut adopter. En cas de compatibilité entre les buts et les plans d'un agent avec ceux du rôle sollicité, un contrat social entre l'organisation et l'agent est établi. Un contrat social est noté $SC = (a, r, CC)$ et comprend les identifiants respectifs de l'agent et du rôle, ainsi que les clauses du contrat. Ces dernières sont établies à partir des normes qui correspondent au rôle (r) adopté par l'agent. La spécification d'une norme a l'une des formes suivantes : $\varphi ::= O_r\varphi | P_r\varphi | F_r\varphi$ où $O_r\varphi, P_r\varphi, F_r\varphi$ désignent respectivement l'obligation, la permission et l'interdiction pour le rôle r de réaliser φ. Les auteurs précisent que des clauses de rupture de contrat peuvent également être spécifiées dans un SC. Mais ils ne disent pas comment les représenter.

La gestion des sorties dans ce modèle dépend donc des clauses de rupture ou fin de contrat définies dans le contrat social de chaque agent.

Le contrôle du comportement des agents au sein d'une organisation est basé sur la vérification des engagements enregistrés dans le contrat de chaque agent. En effet, les clauses d'un contrat constituent un ensemble d'engagements entre un agent vis-à-vis d'une organisation et vice versa. Elles sont établies à partir des normes des rôles d'un agent, éventuellement celles des groupes auxquels appartiennent ces rôles et celles de scènes dans lesquelles l'agent jouera ces rôles.

Discussion

OPERA a de nombreux points communs avec Islander notamment avec la possibilité de spécifier une ontologie et des actes de communication au sein d'une organisation, ainsi que la structuration d'une organisation en scènes. Cependant contrairement à Islander, la spécification des normes avec Opera ne permet pas d'expliciter uniquement des obligations, mais aussi des permissions et des interdictions. En plus, il est possible de définir des normes pour un groupe ou à une scène qui s'appliquent alors à tous les rôles du groupe ou de la scène.

Lors de l'instanciation des normes relatives à un rôle dans un contrat, l'agent de ce contrat peut négocier les termes ce qui représente un élément positif pour les entrées / sorties et le contrôle pour des organisations ouvertes. L'inconvénient de ce modèle est son approche de gestion des entrées des agents. Le fait qu'elle soit basée sur une comparaison entre les buts et plans d'un agent et ceux du rôle qu'il voudrait adopter nous semble enfreindre la privacité de l'architecture des agents. Car ces derniers sont obligés de dévoiler leurs plans et leurs buts avant de pouvoir entrer dans une organisation. D'autre part, il n'existe pas encore d'OMI pour Opera. Il n'est donc pas possible d'étudier la mise en oeuvre des propositions de cet OML et de la confronter à celle d'autres modèles.

3.6.4 Entreprises Virtuelles

Plusieurs approches de modélisation et de gestion d'organisations de type entreprise virtuelle existent dans la littérature [27, 50, 79] avec plus ou moins de ressemblances. Dans cette section, nous présentons celle proposée par Oliveira et al. [32] dont l'OML est d'une bonne expressivité pour la spécification et la gestion des procédures d'entrée / sortie et de contrôle d'organisations ouvertes.

Une entreprise virtuelle est une organisation formée à partir d'un consortium d'entreprises. Chaque entreprise membre du consortium a la responsabilité de réaliser un ou plusieurs buts ou services d'une organisation. Concrètement, une organisation ayant un but global, le décompose en sous-buts à réaliser et en fait des appels d'offres qu'il publie afin que des entreprises ayant les compétences ou capacités appropriées puissent postuler à leurs réalisations.

La procédure de gestion des entrées dans ce type d'organisation est donc basée sur le but à réaliser. A cet effet et contrairement à de nombreux modèles organisationnels, des exigences précises sont spécifiées par l'organisation pour chaque but. Parmi ces exigences, on a le délai de réalisation du but, des propriétés / attributs qui devront caractériser le but à la fin de sa réalisation. Ainsi, une entreprise qui sollicite entrer dans ce type d'organisation doit s'engager à satisfaire les exigences du but qu'elle veut réaliser. Ces exigences impliquent une procédure d'analyse des demandes d'entrée qui a les inputs suivants :

- L'appel d'offre correspondant au but sollicité avec ses exigences et qui est représenté comme suit : $announcement(MAg, GId, D)$ avec *MAg* l'identifiant de l'agent responsable de la gestion de la procédure d'entrée ; *GId* l'identifiant du but à réaliser ; *D* le délai exigé pour la réalisation du but.
- Une proposition de réponse à l'offre : $proposal(OAg, PId, GId, T, LAt)$ avec *OAg* l'identifiant de l'agent représentant l'entreprise qui propose l'offre ; *PId* l'identifiant de la proposition ; *GId* l'identifiant du but à réaliser ; *T* la durée proposée pour la réalisation du but ; *LAt* la liste des attributs exigés dans l'appel d'offre et les valeurs possibles pour chaque attribut, valeurs proposées par l'entreprise.

L'analyse des offres est gérée dans l'OMI par l'agent appelé MarketAgent. Elle consiste en l'étude des adéquations entre les exigences de l'organisation et les données proposées par les entreprises dans leurs offres. A l'issue de l'analyse des offres, si aucune offre ne satisfait totalement l'organisation, celle-ci peut négocier certaines exigences de l'appel d'offre avec les entreprises postulantes. L'objectif étant de trou-

ver des compromis entre des valeurs d'attributs exigées par l'organisation et celles proposées par des offres. À l'issue de l'analyse des offres, un contrat est établi entre l'entreprise retenue et l'organisation selon la représentation suivante :

$$VEContract =< H, CoopEff, BP >$$

- $H(Header) = \langle Id, NormSys, Partics, Ress, Date, Signs \rangle$: l'entête du contrat dans laquelle on a l'identifiant du contrat, le système normatif considéré, les identifiants des parties signataires, l'ensemble des ressources ou services à fournir par les parties, la date et les signatures de chaque partie. Le *système normatif* ici représente le référentiel des normes en vigueur dans l'organisation. Par exemple : les législations Française, Portugaise, Européenne, ... sont des exemple de système normatif pouvant être utilisé dans certaines application.
- $CoopEff(Cooperationeffort) = \{< partic_i, Res_k, Wload >\}$: est un ensemble de triplets. Chaque triplet représente un service Res_k que doit réaliser la partie $partic_i$ selon la charge de travail $Wload =< MinQt, MaxQt, Freq, UnitPr >$.
- $BP(Businessprocess) =< \{ReqPerm_m, ObliChain_n\} >$: est l'ordonnancement des services $Ress$ définis dans l'entête (H). Par exemple, si dans un contrat on a : $Ress = \{S1, S2\}$ avec $S1$ le service que doit fournir l'entreprise et $S2$ la rémunération correspondante à $S1$, on peut avoir un ordonnancement $S1$ doit être réalisé avant $S2$. C'est dans BP qu'on spécifie cet ordonnancement.

L'approche de gestion des entrées proposée dans ce modèle est mieux élaborée que ceux des modèles présentés précédemment. Mais, la gestion des sorties n'est pas abordée par les auteurs.

Quant au contrôle, il est basé sur les clauses de contrat comme dans le modèle Opera. Un engagement de type obligation est représenté comme suit : $Obl_{b,c}(f, d)$ avec b l'agent qui doit satisfaire l'obligation, c l'agent bénéficiaire de l'obligation, f l'objet de l'engagement et d le délai de réalisation de l'engagement. Afin d'avoir une gestion assez flexible du contrôle, l'approche considère qu'à expiration du délai d de réalisation d'un engagement, celui-ci n'est pas automatiquement en état de violation de délai noté $Viol_{b,c}$ tant que l'agent bénéficiaire c ne dénonce pas un état de violation noté $Den_{c,b}(f, d)$. L'engagement reste dans un état appelé *deadline violation* et noté $DViol_{b,c}$ jusqu'à ce qu'il soit satisfait par b, ou que son bénéficiaire dénonce la non satisfaction. On a alors les transitions d'états suivants représentés en logique déontique :

$$Obl_{b,c}(f,d) \wedge (f\ B\ d) \models Fulf_{b,c}(f,d)$$
$$Obl_{b,c}(f,d) \wedge (d\ B\ f) \models DViol_{b,c}(f,d)$$
$$DViol_{b,c}(f,d) \wedge (f\ B\ Den_{c,b}(f,d)) \models Fulf_{b,c}(f,d)$$
$$DViol_{b,c}(f,d) \wedge (Den_{c,b}(f,d)\ B\ f) \models Viol_{b,c}(f,d)$$

La notation $(x\ B\ y)$ se lit x s'est produit avant y, B correspond à *Before*.
Exemples :

- $(f\ B\ d)$ se lit : f est satisfait avant d.
- $(Den_{c,b}(f,d)\ B\ f)$ se lit la dénonciation par c envers b sur son engagement à satisfaire f avant d est faite avant que f soit satisfait.

La procédure de décision dans cette approche de contrôle concerne d'une part l'état noté S d'un engagement, d'autre part la satisfaction de l'objet f. Les auteurs notent $v_a(f)$ et $v_a(S)$ l'évaluation qu'un agent a donne à l'état d'un engagement (S) ou à satisfaction de f. Cette évaluation peut être faite aussi bien par l'agent b qui s'est engagé que par l'agent c, le bénéficiaire de l'engagement. Cette évaluation leur permet de décider des actions à entreprendre. Exemples :

- $v_b(f) < v_b(O_{b,c}(f,d))$: signifie que le coût de réalisation de f est plus élevé que la valeur de f. Une telle évaluation peut entraîner la décision par b de ne pas satisfaire f.
- $v_c(Viol_{b,c}(f,d)) \geq 0$: signifie que la violation d'engagement par b à satisfaire f a un coût supérieur à zéro. Ce qui peut entraîner c à demander un dédommagement à b.

L'approche de contrôle offre d'autres mécanismes très intéressants tel que la distinction de deux types de délai de réalisation, l'un qui est fixe et l'autre qui est compris dans un intervalle. Nous ne pouvons présenter en détail toutes les spécificités de cette approche dans ce mémoire. Le lecteur peut ce référer aux articles [33, 31, 34] pour en savoir plus.

Discussion

L'approche de modélisation et de gestion d'entreprises virtuelles proposée par Oliviera et al. est centrée sur les objectifs d'une organisation. Ainsi, l'OML ne permet pas de structurer des organisations en rôle et autre entité structurelle tels que les groupes, ou les scènes, etc. Cependant, la procédure d'entrée dans une organisation est décrite de façon explicite avec l'avantage de pouvoir exprimer des exigences pour chaque but pour lequel un agent s'engage lorsqu'il entre dans une organisation. La procédure d'entrée offre également aux agents l'avantage de pouvoir négocier les termes de leurs engagements ce qui est intéressant pour l'ouverture. L'approche de contrôle est elle aussi explicitement présentée et est très intéressante par sa flexibilité notamment avec

la spécification de la notion de *violation d'engagement*. Quant à la gestion des sorties, elle est basée comme dans le modèle OPERA sur les clauses établies dans le contrat de chaque agent.

3.6.5 MOISE+ et Moise^{Inst}

Le modèle Moise+ [83] et son extension Moise^{Inst} [68] sont des modèles de spécification d'organisation dont la décomposition structurelle est faite en rôles et en groupes. Un *groupe* est constitué d'un ensemble de *rôles*. Les objectifs d'une organisation sont décomposés en buts et missions. Une *mission* est constitué d'un ensemble de *buts* à réaliser. Les missions sont associées aux rôles par des normes. Ainsi un agent qui entre dans une organisation décrite avec ces modèles y jouera un ou plusieurs rôle(s) et devra réaliser les missions associées à ceux-ci.

La procédure d'entrée dans une organisation pour ces modèles est basée sur l'*adoption de rôle*. Par ailleurs, malgré le fait que des missions soient associées à des rôles, aucune exigence n'est spécifiée pour l'adoption d'un rôle. L'analyse d'une demande d'entrée est gérée selon le modèle par un OrgManager pour Moise+, un InstManager et un StructManager pour Moise^{Inst}. Elle consiste essentiellement à la vérification de la cardinalité maximale du rôle sollicité pour l'entrée. Comme outputs, dans les deux modèles, l'agent reçoit un proxy appelé *OrgBox* dans Moise+ et *OrgWrapper* dans Moise^{Inst}. Il devient également GroupMember du groupe auquel appartient le rôle qu'il a adopté et implicitement un OrgMember.

La sortie d'un agent d'une organisation nécessite un désengagement de celui-ci à toutes les missions de tous les rôles qu'il a adopté dans l'organisation et un abandon explicite de chacun de ces rôle(s). Dans ces modèles, l'abandon d'un rôle par un agent ou la sortie d'un agent d'une OE peut nécessiter une réorganisation qui est gérée de façon réflexive par une spécification de procédure de réorganisation avec l'OML [82].

La gestion du contrôle dans les deux modèles est basée sur leur spécification de norme. Dans Moise+, une norme est représentée comme suit : $r\ \alpha\ m$ avec r un rôle, m une mission et α une relation déontique pouvant être une obligation, une permission ou une interdiction. Le contrôle dans ce modèle est ad-hoc et rigide car un agent ne peut enfreindre une norme définie dans une organisation.

Dans le modèle Moise^{Inst} l'intégration d'une dimension contextuelle a contribué à enrichir la spécification d'une norme qui est représentée de la façon suivante :

$$n = \rho \rightarrow op(cont, issuer, bearer, m, sanc, w, tc).$$

- ρ : est la condition d'activation de la norme.
- op : est l'opérateur déontique dont les valeurs peuvent être *obl,per,for*.
- $cont$: est le contexte dans lequel s'applique la norme.
- $issuer$ et $bearer$: sont des entités structurelles (rôle, groupe) représentant respectivement l'émetteur de la norme et le porteur de la norme.
- m : est la mission sur laquelle porte la norme.
- $sanc$: est une sanction applicable en cas de violation de la norme.
- w est la pondération de la norme utile en cas de conflit entre normes.
- tc est la durée de validité de la norme après qu'elle soit active.

Cette spécification apporte une modération à la gestion du contrôle au sein des organisations. En effet, les paramètres $sanc$ et w expriment explicitement les conséquences correspondantes respectivement au respect ou non d'une norme par un agent. Les sanctions sont appliquées au sein de l'organisation par un NormManager. Enfin, précisons que le modèle Moise^{Inst} offre également la possibilité de spécifier de façon explicite les règles de contrôle ou supervision d'une organisation. Pour cela elle utilise de façon réflexive l'OML. Plus précisément, les missions et buts correspondants à chacun des rôles InstManager, StructManager, FunctManager et NormManager peuvent être définis dans une spécification fonctionnelle de supervision. Chacun de ces rôles étant préalablement défini dans une spécification structurelle. Les contextes d'application des règles de contrôle peuvent être définis dans la spécification contextuelle et les normes qui associent rôle, mission, contexte, ... dans une spécification normative.

Discussion

Les différentes versions de ce modèle, offrent des bases intéressantes de spécification et gestion d'organisations ouvertes. En effet, la décomposition très explicite d'une organisation en dimensions introduite par l'OML Moise+ et approfondies par Moise^{Inst}, favorise la modularisation de la gestion des fonctionnalités telles que l'échange de données avec l'environnement, les entrées / sorties et le contrôle.

Par ailleurs, contrairement au modèle Opera et celui des entreprises virtuelles, aucune des versions de Moise ne permet pas de spécifier les compétences nécessaires à l'entrée dans une organisation. Ainsi tout agent peut prétendre à la candidature d'un rôle même s'il n'a pas les capacités ou compétences requises.

3.7 Synthèse

Nous avons présenté dans ce chapitre l'intérêt des organisations pour les SMA relativement à l'ouverture (cf. section 3.1). Nous avons ainsi pu mettre en avant leurs deux principales caractéristiques à savoir :

- La modularisation qui permet de décomposer structurellement et fonctionnellement des applications et en conséquence de mieux appréhender leur complexité.
- La régulation qui permet de spécifier des normes qui contraignent l'autonomie des agents en décrivant les comportements qu'ils devront avoir au sein d'une organisation.

Nous avons ensuite présenté les outils que les modèles organisationnels utilisent d'une part pour la représentation de la modularité et des règles de régulation et d'autre part pour la gestion proprement dite des instances d'organisations. Ainsi, les modèles organisationnels multi-agents utilisent un *langage de modélisation d'organisations* (OML) pour représenter une organisation en différentes dimensions : structurelle, dialogique, fonctionnelle, contextuelle, normative. Tous les modèles existant n'offrent pas toutes les dimensions suscitées. Cependant quel que soit le nombre de dimensions considérées par l'OML d'un modèle, la description d'une organisation avec cet OML constitue ce qu'on appelle une *spécification organisationnelle* (OS). Cette dernière joue un rôle très important pour l'ouverture notamment pour la propriété d'interopérabilité.

Les organisations spécifiées avec un OML sont gérées par une plate-forme appelée *infrastructure de gestion d'organisation* (OMI) qui implémente les mécanismes de gestion relatives à cet OML. Une OMI permet de gérer les différents services relatifs au fonctionnement d'instances organisations appelées *entités organisationnelles* (OE). Les services gérés par une OMI sont :

- Les échanges avec les agents extérieurs qui concernent principalement la gestion de procédures d'entrée d'agents.
- Les activités des agents après leur entrée au sein d'une organisation. La gestion des activités comprend la coordination (gestion des coopérations), la régulation des comportements et la gestion des sorties.

Nous avons enfin étudié quelques modèles organisationnels relativement à leur mise en oeuvre concrète des propriétés d'ouverture.

Synthèse interopérabilité des modèles organisationnels

Les échanges de données entre une organisation et des agents extérieurs à celle-ci sont l'OS et la ou les procédure(s) d'entrée / sortie dans l'organisation. Dans la section 3.2, nous avons présenté la notion d'OML qui est le langage que les modèles organisationnels fournissent pour définir des OS avec plus ou moins d'expressivité (cf. tableau 3.1). Ainsi, chaque modèle organisationnel a un OML avec une description plus ou moins formelle. Des agents peuvent donc accéder à tout ou partie des OS d'organisations définies avec ces OML. Cependant, nous pouvons constater à travers le tableau 3.1 qu'il n'existe pas de dimension spécifique à la spécification de procédure d'entrée / sortie dans des organisations. En effet, ces procédures sont généralement décrites de façon textuelle et pas très explicite.

En ce qui concerne les interfaces d'interactions d'une organisation qui permettent les échanges de données à disposition d'agents extérieurs à l'environnement de l'organisation, elles varient d'un modèle organisationnel à un autre. De façon générale, les données échangées sont transmises par des messages. En effet, les organisations sont représentées et gérées dans une OMI par un ou plusieurs *agent(s) manager* –InstitutionManager, SceneManager [57], OrgManager [83], StructManager, FunctionnalManager [68], etc– qui traitent les demandes et actions des agents. Ainsi, les interactions entre une organisation et un agent sont de type *communication entre agents*. En conséquence, l'utilisation de la spécification FIPA ACL par certaines OMI permet d'avoir une interopérabilité syntaxique des messages échangés. D'autre part, nous avons vu dans la section 3.3 que les OMI des modèles organisationnels offrent généralement des proxies qui assurent la médiation des interactions entre agents et organisation. Ainsi, les interfaces d'échanges entre les organisations et leur environnement sont les primitives organisationnelles et actes de langages que gèrent les proxies des OMI.

En ce qui concerne l'interopérabilité sémantique, s'il est vrai que presque tous les modèles organisationnels utilisent le standard XML pour représenter les OS, l'interprétation sémantique de ces derniers n'est pas garantie. En effet, il n'existe pas d'ontologie SMA qui définit les concepts en particulier les concepts organisationnels tels que rôle, groupe, scène, tâche, but, mission, norme, ... Ainsi chacun de ces concepts peut avoir une sémantique différente d'un OML à un autre ce qui ne favorise pas une interprétation sémantique unique des données échangées au cours des interactions entre organisation et agents. L'interopérabilité sémantique reste

donc une problématique ouverte [41] pour des organisations multi-agents ainsi que pour la communauté SMA en générale car, la spécification et l'adoption de standards ontologiques relèvent plus des objectifs d'une communauté que de celles de recherches individuelles.

En résumé, en considérant les trois aspects d'interopérabilité : structurelle, syntaxique et sémantique [40, 63], nous pouvons dire que l'interopérabilité syntaxique est celle qui est la plus mise en oeuvre dans les modèles organisationnels de la littérature. Car, ils utilisent généralement le standard XML pour décrire leur OML. Beaucoup de modèles ne fournissent pas de spécification organisationnelle explicite ainsi que des interfaces d'interactions bien décrites qui facilitent les échanges entre une organisation et des agents extérieurs. Ces modèles négligent ainsi l'interopérabilité structurelle. Quant à l'interopérabilité sémantique elle reste une problématique ouverte.

Synthèse entrée / sortie des modèles organisationnels

L'étude des modèles que nous avons présentés dans les sections 3.5 et 3.6 nous a permis de constater une récurrence de l'utilisation de la notion de *rôle* comme moyen d'entrée dans une organisation(cf. 3.2). En effet, parmi les modèles étudiés, seuls deux modèles RIO [123] et les entreprises virtuelles [32] utilisent des concepts différents à savoir *agent abstrait* pour RIO et *but* pour les entreprises virtuelles. En outre, la notion d'agent abstrait peut être considérée comme un synonyme de celle de rôle. On peut donc dire que de façon distincte, on a deux concepts différents : *rôle* et *but*. D'autre part, dans le modèle AGRS [118], le concept de *groupe* est également proposé comme moyen d'entrée dans une organisation. Mais une fois entré, l'agent doit jouer un rôle pour pouvoir offrir où utiliser un service au sein de l'organisation. Dans le modèle Islander [56], l'entrée dans une institution électronique (EI) se fait en adoptant également un rôle dans la *scène initiale* de l'EI.

Enfin, que le concept représentant le moyen d'entrée soit le rôle ou le but, il représente soit une position des agents dans l'organisation et donc implicitement des responsabilités –tâches / services / missions à réaliser–, soit plus directement avec le concept de but, des objectifs à atteindre et donc nécessite une intégration de l'agent dans l'entité organisationnelle.

Concernant justement l'intégration des agents admis dans une OE, nous avons remarqué que l'approche la plus utilisée consiste à fournir un proxy à chaque agent. On

dénombre une grande variété de concepts représentant les proxies. En effet chaque modèle utilise un concept différent : *compétence, AgentInRole, TeamcoreProxy, Governor, OrgBox, etc.* (cf. Tab 3.2). Par contre, certains modèles ont des approches assez originales de gestion de l'intégration des agents (cf. Tab 3.2). C'est le cas par exemple de BRAIN qui propose une fusion du code de chaque agent avec celui du rôle qu'il adopte au sein d'une organisation. C'est également le cas de l'approche proposée par Oliveira et al. avec l'utilisation de contrat dans lequel sont enregistrés les engagements d'un agent vis-à-vis de l'organisation dans laquelle il entre ainsi que les engagements de l'organisation vis-à-vis de l'agent. Cette proposition est très intéressante dans la mesure où elle permet aux agents de négocier les engagements de leur contrat, aussi bien les leurs que ceux de l'organisation avec laquelle ils signent un contrat. De plus cette approche implique une gestion différente des contrôles qui sont alors basés sur les engagements d'un contrat et pas uniquement sur la spécification des normes.

Suivant le modèle considéré, des exigences peuvent être spécifiées de façon plus ou moins explicite pour le concept du point d'entrée. En effet, nous avons remarqué que très peu de modèles, permettent de spécifier des exigences pour les demandes d'entrées. Seuls les modèles ROAD [39], OPERA [52], Islander [56] AGRS [118] et EV [32] prennent en compte la notion d'exigences dans leurs processus de gestion des entrées. Concrètement seuls les modèles OPERA et EV proposent une description et représentation explicite des exigences. Les modèles ROAD, Islander et AGRS en parlent mais sans préciser comment elles sont représentées.

Parmi les modèles étudiés, très peu proposent une approche de gestion des sorties encore moins avec une spécification explicite. En effet, les modèles évoquent plus souvent l'approche de gestion de l'abandon d'un rôle qui consiste en l'abandon du proxy associée : *certificat, compétence, AgentInRole, ...* Nous avons cependant noté deux approches intéressantes celle de la gestion de la vacance d'un rôle proposée par les modèles RIO et ROAD, ainsi que celle basée sur la vérification de conditions relatives à la sortie préalablement enregistré dans le contrat de l'agent proposée par THOMAS [6], OPERA, et EV.

En outre, presque tous les modèles ont un seul point d'entrée et donc une approche de gestion centralisée.

Synthèse contrôle dans les modèles organisationnels

Deux principales approches de gestion des contrôles ressortent de notre étude des modèles présentés dans les sections 3.5 et 3.6.

L'approche des modèles non normatifs qui est rigoureuse. En effet, des normes ne

pouvant être spécifiées avec ces modèles, le contrôle est implémenté de façon ad-hoc. Dans des organisations définies avec ce tye de modèle, les agents n'ont donc pas la possibilité d'avoir des comportements différents à la spécification organisationnelle. Avec les modèles normatifs, les comportements des agents sont régis par des normes. Nous avons remarqué que pour certains modèles de ce type, l'approche de contrôle est la même que pour les modèles non normatifs. C'est le cas des modèles Islander, Moise+, AGRS, En effet, les normes permettent dans ces modèles de définir des comportements quasiment obligatoires d'agents. Pour garantir le respect des normes, les règles de contrôle sont implémentés de façon à ce que les agents ne puissent pas les enfreindre. Cette approche de contrôle des normes est relativement la même que celle de l'approche de contrôle d'accès basé sur les rôles (RBAC) présentée dans la section 2.3. En effet, avec l'approche RBAC, des permissions sont attribuées à des utilisateurs d'un système. Les mécanismes de contrôle sont implémentés de façon à ne pas permettre aux utilisateurs de réaliser des actions qui ne sont pas autorisées par les permissions associées à leur rôle.

La deuxième approche de contrôle identifiée dans notre étude des modèles est celle qui est un peu flexible proposée par le modèle EV. Elle comprend une phase de constat de la violation d'une norme et une phase intermédiaire de décision. Dans cette dernière, soit l'agent qui est en violation de norme peut faire changer l'état de celle-ci en réalisant l'engagement concerné, soit l'agent contrôleur de la norme peut faire une dénonciation effective de violation de norme ce qui pourrait entraîner une sanction.

Grille de lecture du tableau 3.2

Le tableau 3.2 résume cette synthèse en présentant pour chaque modèle organisationnel étudié, les niveaux de prise en compte et quelques outils de mise en oeuvre des propriétés d'interopérabilité, d'entrée/sortie et de contrôle.

Pour la propriété d'*interopérabilité*, les signes +, ++ et +++ catégorisant l'aspect *syntaxique* de cette propriété pour chaque modèle doivent être interprétés de la façon suivante :

1. + : le modèle offre une spécification organisationnelle (OS) avec description des procédures d'entrée / sortie non formelle, essentiellement textuelle.
2. ++ : le modèle offre une OS avec description des procédures d'entrée / sortie formelle de moindre expressivité
3. +++ : le modèle offre une OS avec description des procédures d'entrée / sortie formelle de bonne expressivité.

La catégorisation des aspects *structurel* et *sémantique* de l'interopérabilité dans notre tableau récapitulatif se fait essentiellement avec les signes + et - qui doivent être interprété de la façon suivante :

1. Mise en oeuvre de l'interopérabilité structurelle : (+) si le modèle offre des interfaces d'échanges de données entre l'organisation et les agents de son environnement ; (-) sinon.
2. Mise en oeuvre de l'interopérabilité structurelle : (+) si le modèle offre une pseudo-ontologie pour les communications entre agents ; (-) sinon.

Les colonnes correspondantes au récapitulatif de la mise en oeuvre de la propriété de gestion des *entrées / sorties* présentent respectivement :

1. Le moyen défini par le modèle pour permettre aux agents d'entrée dans une entité organisationnelle (OE).
2. Le moyen d'intégration d'un agent admis dans une OE.
3. L'existence d'une procédure de sortie d'une OE. Le signe (+) exprime l'existence d'une procédure simple de gestion des sorties –parfois assimilée à l'abandon d'un rôle– décrite de façon textuelle ou speudo-formelle par les clauses de contrat. (++) exprime l'existence d'une description textuelle de procédure originale de gestion de vacance ou d'abandon d'un rôle, sans description explicite de procédure de gestion de sortie.
4. Le type de gestion des entrées / sorties : *simple* ou *originale*.

Enfin, le résumé de l'étude de la propriété de *contrôle* présenté dans le tableau concerne respectivement :

1. L'existence ou pas d'un processus explicite de contrôle dans un modèle. Le signe (–) exprime le fait que le processus n'est pas décrit ; (+) exprime une description essentiellement textuelle du processus ; (++) exprime une description formalisée du processus et (+++) exprime une description formalisée et et un processus originale.
2. Le type d'approche de gestion du contrôle par un modèle : avec le signe (–) pour les approches non identifiées et dans le cas contraire, les types *Rigide* et *Flexible* selon l'approche utilisée dans un modèle.

Modèles		Interopérabilité			Entrée / Sortie				Contrôle	
		Syn-taxique	**Struc-turelle**	**Séman-tique**	**Moyen entrée**	**Intégration agents**	**Procédure sortie**	**Type gestion**	**Processus explicite**	**Type gestion**
Non normatifs	RIO [123]	++	+	+	Agent abstrait	Compétence	+	Simple	–	–
	ROAD [39]	+	+	+	Rôle	Contrat	+	Originale	++	–
	BRAIN [25]	+	+	–	Rôle	Certificat fusion codes	+	Simple	–	–
	AGRS [118]	++	+	–	Rôle Groupe	AgentInRole	+	Simple	–	–
	Teamcore [112]	++	+	–	Rôle	Teamcore proxy	+	Originale	+	Flexible
Normatifs	THOMAS [6]	++	+	–	Rôle	Services	++	Simple	+	Rigide
	Islander [56]	++	+	+	Rôle	Governor	+	Simple	+	Rigide
	OperA [52]	++	–	+	Rôle	Contrat social	++	Originale	+	–
	EV [32]	+++	+	+	But	Contrat	++	Originale	+++	Flexible
	Moise [81]	++	+	–	Rôle	OrgBox	+	Simple	+	Rigide
	Moise^{Inst} [68]	++	+	–	Rôle	OrgWrapper	++	Simple	++	Rigide

TABLE 3.2 – Récapitulatif de la prise en compte des propriétés d'entrée sortie par quelques modèles organisationnels

Chapitre 4

Synthèse première partie

Notre problématique dans cette thèse est d'étudier et proposer des outils qui favorisent la gestion de l'ouverture dans des organisations multi-agents. Cette problématique s'impose de plus en plus dans le développement de systèmes modernes dont les besoins d'interactions avec d'autres systèmes ayant des fonctionnalités complémentaires sont de plus en plus croissants.

Pour ce faire, nous avons dans cette première partie du mémoire analysé l'état de l'art de la problématique d'ouverture dans le cadre très général des systèmes informatiques ainsi que dans le cadre des systèmes multi-agents.

L'étude de l'ouverture dans les systèmes informatiques nous a permis d'identifier trois propriétés caractéristiques de systèmes ouverts :

1. L'*interopérabilité* : qui concerne les échanges de données entre un système et son environnement. L'objectif pour un système ouvert grâce à cette propriété est d'être capable de recevoir et utiliser les inputs fournis par les autres systèmes de son environnement et pour ces derniers d'être capables de recevoir et utiliser les outputs que leur fournit le système avec lequel ils interagissent.
2. Les *entrées / sorties* d'entités : qui concernent l'extension et la restriction de la frontière d'un système. La frontière d'un système détermine l'appartenance ou pas d'une entité –ressources matériels, logiciels ou humains– au système. Des entités appartenant à un système se distinguent de celles qui ne lui appartiennent pas par le fait que les premières sont intégrées dans l'architecture structurelle et fonctionnelle du système et sont sous son contrôle, ce qui n'est pas le cas des secondes. L'appartenance d'une entité à un système passe par un processus d'entrée et d'intégration. La propriété de sortie résulte de la possibilité d'une

part pour les entités d'un système à le quitter, d'autre part pour un système à se séparer de certaines de ces entités.

3. Le contrôle : est une propriété qui résulte de la sensibilité des données manipulées par les systèmes informatiques lors de leurs échanges avec les entités de l'environnement ou de leur traitement au sein du système.

La mise en oeuvre des deux premières propriétés est généralement basée sur des standards définis par des organismes de normalisation. Ces derniers spécifient pour différentes fonctionnalités les types d'entités intervenant dans les interactions et échanges de données, les types et formats de données échangées, les interfaces d'échanges et d'interactions qui contribuent à la gestion de ces propriétés dans un système. La propriété de contrôle quant à elle est abordée dans les problématiques de sécurité. Cependant, l'ambition d'avoir des standards pour toutes les applications informatiques et pour chacune d'elle pour toutes leurs fonctionnalités n'est pas réalisable. Raison pour laquelle nous avons vu qu'à défaut de pouvoir utiliser des standards, des concepteurs de systèmes ouverts doivent offrir aux entités d'environnement une description des fonctionnalités de leurs systèmes, des interfaces d'interactions et des données échangeables.

Afin d'approfondir l'étude de la mise en oeuvre de l'ouverture dans des systèmes informatiques, nous avons étudié l'approche RBAC (role based access control) [61, 120] de gestion de l'ouverture basée sur le contrôle d'accès à un système. De cette étude, nous avons pu noter que les entités de l'environnement avec lesquelles un système a des échanges de données sont des utilisateurs humains. Les interfaces d'échanges de données dépendent des systèmes dans lesquels est développée une spécification RBAC. La gestion des entrées / sorties est basée sur l'attribution de rôles aux utilisateurs et l'abandon de rôles par des utilisateurs. L'intégration d'un nouvel utilisateur dans un système dépend donc de son rôle. Le contrôle est basé sur les permissions associées aux primitives sous-jacentes à la gestion des entrées / sorties et du contrôle sont spécifiées par la norme INCITS 359-2004 [62]. rôles d'un système.

Nous avons vu que l'approche RBAC offre l'avantage de structurer des utilisateurs d'un système avec le concept de rôle. Ce dernier permet également une gestion agrégée des droits d'accès puisque les droits d'accès ne sont pas gérés individuellement pour chaque utilisateur, mais pour tous les utilisateurs ayant un même rôle. Les concepts de *rôle* et de *permission* utilisés dans RBAC constituent des éléments de ressemblance de cette approche avec les organisations en particulier les organisations multi-agents que nous avons étudiées dans le chapitre 2. La spécialisation du prin-

cipe de base dans le modèle Or-Bac [3] illustre d'avantage cette ressemblance avec la contextualisation des droits d'accès qui est comparable à la dimension contextuelle des modèles organisationnels multi-agents.

Comme limites de l'approche RBAC pour la gestion de l'ouverture, nous avons déjà mentionné dans le chapitre 2 le fait qu'il n'est pas possible de spécifier des exigences relatives aux attentes d'un système vis-à-vis des utilisateurs pour le processus d'attribution de rôle. Par ailleurs, la rigidité du contrôle dans les modèles RBAC liée au fait qu'un utilisateur ne peut pas enfreindre les permissions du (ou des) rôle(s) qui lui est (ou sont) attribué(s) peut ne pas être appropriée pour certaines applications.

Après avoir étudié l'ouverture dans les systèmes informatiques et l'approche de mise en oeuvre proposée par RBAC, nous nous sommes intéressés à la façon dont l'ouverture est intégrée dans les travaux du domaine des systèmes multi-agents. Les SMA se distinguent des systèmes informatiques classiques par le fait qu'ils sont constitués d'entités autonomes réactives ou cognitives appelées agents [138, 9]. Nous avons pu constater que les principales caractéristiques des SMA à savoir l'*hétérogénéité*, l'*autonomie* des agents ainsi que leurs *coopérations* impliquent une mise en oeuvre particulière des propriétés d'ouverture dans ces systèmes.

- L'*hétérogénéité* des agents est liée au fait que dans un SMA, les agents peuvent être de langages d'implémentations différents (Jade [15], Jason [22], Jadex [4], etc.), d'architectures de raisonnements différents (BDI [114], réactif [36], etc.).
- L'*autonomie* des agents est la propriété qui distingue un agent d'un programme informatique classique dans la mesure ou un agent a la capacité de décider de ses actions à réaliser ou non sans l'intervention d'un tiers (humain ou autre programme). Tandis qu'un programme informatique classique réalise une action suite à l'exécution par une entité extérieure (humain ou autre programme) d'une instruction appropriée.
- La *coopération* entre les agents d'un SMA est liée au fait qu'en général, aucun agent ne dispose de toutes les capacités nécessaires à la résolution d'un problème complexe.

En tenant compte des caractéristiques des SMA ci-dessus citées, la mise en oeuvre des propriétés d'ouverture a impliquée d'une part, la spécification d'un standard de langage de communication entre agents (FIPA ACL [64]) pour faciliter les échanges et interactions entre agents hétérogènes. D'autre part, la spécification d'une architecture standard de SMA avec un service de pages jaunes [64] a également été définie

afin de faciliter la gestion des entrées / sorties d'agents dans un SMA pour des coopérations. La propriété de contrôle quant à elle est abordée par diverses approches dont les principales sont basées sur la spécification de normes (*obligations, permissions et interdictions*) afin de contraindre l'autonomie des agents au sein d'un SMA.

L'ensemble de ces moyens : langage de communication FIPA-ACL, service de pages jaunes, approches de contrôles visent à caractériser un SMA ouvert comme étant un SMA dans lequel des agents hétérogènes, autonomes potentiellement de localisations différentes peuvent y entrer pour des objectifs individuels ou pour des objectifs sociaux et interagir avec d'autres agents.

Comme avantages des mécanismes de gestion de l'ouverture dans les systèmes multi-agents, nous pouvons dire que les standards utilisés pour la gestion des communications des agents et des entrées / sorties, constituent des bases de recherche consistantes pour le développement d'autres problématiques telles que celle d'environnement multi-agent ouvert [104, 136] ou celle d'organisation multi-agent ouvertes [20, 87]. Cependant, les approches de contrôle proposées notamment celles des *systèmes normatifs* [18, 37] constituent encore des problématiques de recherches très actives qui s'intègrent généralement dans les travaux sur les modèles organisationnels que nous avons étudiés dans le chapitre 3.

L'étude des travaux développés sur les organisations dans les SMA nous a permis de mettre en évidence les avantages qu'ils offrent pour le développement d'applications multi-agents [74].

En effet, la modélisation d'une application avec un *langage de modélisation d'organisations (OML)*, permet d'expliciter avec plus ou moins d'expressivité (cf. tableau 3.1) les données structurelle, fonctionnelle, contextuelle, dialogique et normative des applications. Ainsi, la modélisation permet de mieux appréhender la complexité des application et le cas échéant de mieux gérer les modifications et adaptations (réorganisation) de celles-ci. La gestion effective d'applications organisationnelles se fait par des *infrastructures de gestion d'organisations OMI*. Ces derniers assurent différents services (création d'OE, gestion des entrées d'agents et leur intégration, gestion des coopérations et coordinations, gestion des sorties et de la régulation) pour le bon fonctionnement des applications.

En outre, l'étude individuelle de quelques modèles organisationnels (cf. chapitre 3) montre une divergences d'approches proposées, plus ou moins intéressantes et pertinentes pour une réelle modélisation et gestion d'organisations ouvertes.

Nous avons particulièrement remarqué que de nombreux modèles n'offrent pas la

possibilité par leur OML de spécifier explicitement des procédures d'entrée / sortie dans des organisations. D'autre part, de nombreuses infrastructures de gestions d'organisations (OMI) n'offrent pas non plus une approche pertinente de gestion des interactions entre agents et organisations de manière à mieux favoriser l'interopérabilité, les entrées / sorties et le contrôle au sein d'organisations ouvertes.

C'est donc à ces limites que nous nous sommes intéressés dans notre proposition de modèle de gestion d'organisations multi-agents ouvertes présentées dans la prochaine partie de ce mémoire. Ainsi, nous présenterons dans les chapitres 5 et 6 respectivement le langage de modélisation d'organisations ouvertes ainsi que l'approche de gestion d'organisations ouvertes que nous proposons. L'OML que nous proposons est le résultat de l'extension de la richesse des OML Moise+ et Moise^{Inst} avec une dimension de spécification explicite de procédure d'entrée / sortie dans des organisations ouvertes. L'approche de gestion des propriétés d'ouverture que nous présentons dans le chapitre 6 décrit principalement les étapes de gestion des procédures d'entrée / sortie et de régulation des agents au sein d'une organisation ouverte. Dans le chapitres 7, nous présentons l'OMI que nous avons développé pour assurer les fonctionnalités de gestion des propriétés d'ouverture au sein d'une organisation. Cette OMI est basée sur la notion d'*artefacts organisationnels* [89] qui s'inspire de la notion d'*artefact* [104] proposée par A. Ricci et Al. dans leurs travaux sur le méta-modèle A& A (Agents & Artifacts) [117, 105] dont nous présentons quelques spécificités dans le chapitre 7.

Deuxième partie

Modèle de gestion d'organisations multi-agents ouvertes

Chapitre 5

Langage de modélisation d'organisation ouverte : OML Moise

L'étude des modèles organisationnels que nous avons présentée dans le chapitre 3 nous a permis de voir en quoi les organisations constituent un moyen d'ouverture des SMA à travers leur modélisation avec des langages de modélisation d'organisation (OML) et leur gestion avec des infrastructures de gestion d'organisation (OMI). Parmi les OML étudiés, nous avons brièvement présenté les Modèles $\mathcal{M}$OISE^{+} et $\mathcal{M}$OISE^{Inst} sur lesquels sont basés nos travaux de recherche. Ces modèles ont l'avantage de permettre une représentation des organisations en dimensions modulaires qui favorisent la gestion de l'ouverture. Cependant, comme la majorité des modèles organisationnels, ces modèles n'offrent pas de spécification explicite de procédure d'entrée / sortie dans des organisations.

Dans ce chapitre, nous allons donc présenter le langage de modélisation d'organisation MOISE, qui intègre les dimensions structurelle, fonctionnelle, et normative proposées par les modèles $\mathcal{M}$OISE^{+} [81] et $\mathcal{M}$OISE^{Inst} [68], enrichies de nos modifications. La présentation de ces dimensions sera suivie de celle de la dimension d'entrée / sortie (EES : Entry-Exit Specification) que nous proposons. Précisons que l'ensemble des figures présentant les différents éléments du modèle sont en anglais.

5.1 Méta-modèle MOISE

Nous avons défini une organisation multi-agent comme un ensemble d'agents qui interagissent suivant une structure, des schémas de coopération et des normes prédéfinies pour atteindre des objectifs précis. Un OML est un langage qui permet de repré-

senter les données relatives à une organisation telles que : ses objectifs, sa structure, ses schémas de coopération, ses normes. Nous avons également vu que cette représentation est décomposée en plusieurs dimensions dans les modèles organisationnels.

L'OML MOISE dont nous présentons les quatre dimensions dans les sections ci-dessous hérite de la richesse des versions précédentes de ce modèle et offre de nouvelles possibilités d'expressivité pour la modélisation d'organisations ouvertes notamment pour la propriété d'entrée / sortie. En effet, cette nouvelle version conserve et enrichit les spécifications structurelle (SS), fonctionnelle (FS) et normative (NS) définies dans les versions précédentes. Nous lui avons ajouté une notion d'*exigence* qui a été intégrée au concept de rôle d'une SS et aux concepts de mission et de but d'une FS afin d'améliorer les entrées / sorties dans une organisation, les adoptions / abandons de rôle, les engagements / abandons à des missions au sein d'une organisation. Par ailleurs, nous avons défini une nouvelle dimension appelée *spécification d'entrée / sortie* (EES [1]) qui permet de spécifier de façon explicite les procédures d'entrée et de sortie d'une organisation.

En résumé, le méta-modèle de l'OML MOISE que nous proposons dans cette thèse comprend quatre dimensions : structurelle, fonctionnelle, normative et entrée / sortie. Celles-ci sont organisées dans des packages présentés sur la figure 5.1.

1. La dimension structurelle, représentée sur la figure 5.1 par le package *Structural Specification*, permet de spécifier les groupes et les rôles qui définissent la structure d'une organisation. Nous présenterons en détail les éléments de cette dimension dans la section 5.3.
2. Le package *Functional Specification* représente les éléments de la dimension fonctionnelle. Cette dernière permet de spécifier les schémas sociaux qui définissent les coopérations des activités fonctionnelles au sein d'une organisation. Les schémas sociaux sont constitués de buts et de missions que nous présenterons en détail dans la section 5.4.
3. Le package *Normative Specification*, correspond à la dimension normative où sont spécifiées les normes qui établissent les contraintes sous formes d'obligations de permissions et interdictions entre les rôles définis dans la SS et les missions définies dans la FS. Nous le présentons en détail dans la section 5.5.
4. Les éléments du package *Commun* sont des éléments abstraits qui sont spécialisés dans les quatre dimensions du modèle. Ainsi, une entité (Entity) est un élément abstrait pouvant avoir des exigences de type adoption (AdoptReq) et des exigences de type abandon (LeaveReq). Ces deux types d'exigences sont

1. EES : Entry Exit Specification

eux-mêmes des spécialisations de la notion abstraite d'exigence (Requirement) que nous présentons en détail dans la section 5.2. Une entité est spécialisée dans la SS avec la notion de rôle et dans la FS avec les notions de mission et de but.

5. Enfin, la dimension entrée / sortie décrite dans le package EntryExit Specification permet de spécifier les portes d'entrée (*Gate*) et les portails (*portal*) qui gères les entrées et les sorties dans une organisation. Dans cette dimension, sont également spécifiées des exigences d'entrée / sortie (EntryReq, ExitReq) qui sont des spécialisations de *Requirement* du package *Commun*. Nous présenterons cette dimension dans la section 5.6.

La spécification organisationnelle d'une organisation définie avec l'OML MOISE est donc représentée par le le quadruplet suivant :

$$OS = \langle SS, FS, NS, EES \rangle$$

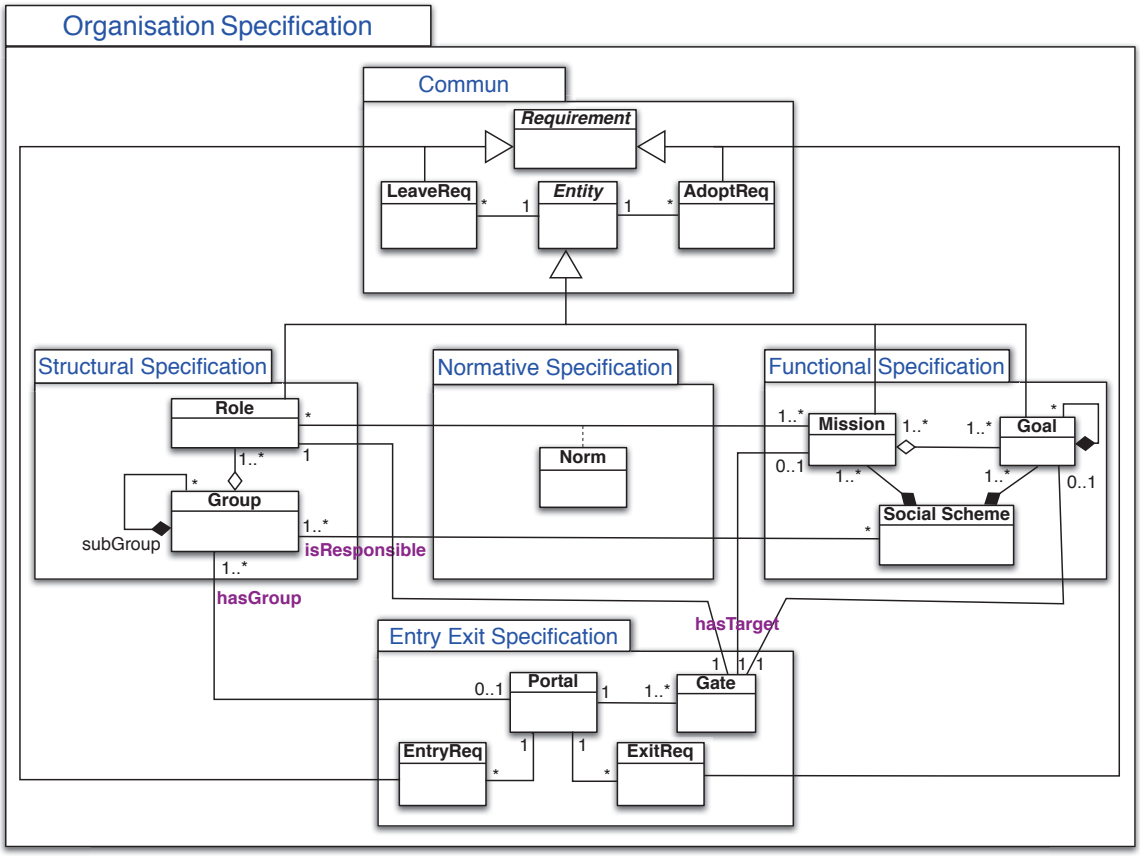


FIGURE 5.1 – Méta-Modèle de l'OML MOISE

L'OML MOISE comme ses versions précédentes aborde plusieurs niveaux – *individuel, social et collectif*– à travers son OS. Particulièrement, les rôles spécifiés dans la dimensions structurelle (SS) définissent un niveau *individuel* relatif aux agents. De plus, dans une SS, des *relations* peuvent être spécifiées entre des rôles (cf. section 5.3.2). Ces relations définissent le niveau *social* qui existe entre des agents qui jouent des rôles dans une organisation. Les groupes auxquels appartiennent des rôles représentent le niveau *collectif* de structuration [68].

Dans une FS les *buts* définissent le niveau *individuel* de fonctionnement d'une organisation c'est-à-dire les objectifs qui devront être atteints individuellement par des agents en fonction de leurs rôles. Les buts sont regroupés en *missions* qui représentent le niveau *social* car une mission peut être attribuée à plusieurs agents qui devront alors se coordonner pour la réaliser. Enfin, les *schémas sociaux* représentent l'organisation fonctionnelle collective d'une organisation et donc le niveau *collectif* d'interprétation de la FS [68].

Nous reviendrons plus en détail sur ces différents niveaux pour la SS et la FS dans les sections 5.3 et 5.4.

Les autres dimensions : normative (NS) et entrée / sortie (EES) permettent quant à elles de définir respectivement les normes qui établissent les comportements que l'organisation attend de la part des agents, et les procédures d'entrée et de sortie, qui permettront de gérer les admissions et intégrations d'agents au sein des entités organisationnelles ainsi que leurs départs.

Comme pour les modèles $\mathcal{M}\text{OISE}^{+}$ et $\mathcal{M}\text{OISE}^{Inst}$, les dimensions d'une spécification organisationnelle proposées peuvent être représentées sous plusieurs formalismes :

- Un formalisme graphique qui offre une représentation visuelle facilement compréhensible et manipulable par des concepteurs d'organisations. Les éléments de représentation graphique des dimensions structurelle et fonctionnelle sont présentés sur les figures 5.2 et 5.4.
- Un formalisme BNF pour une représentation plus formelle des spécifications et des relations qui existent entre elles, et pour chacune d'elles entre leurs éléments [68].
- Pour leur programmation des OMI, un formalisme XML [2] et un formalisme logique (NOPL [78]) ont également été définis.

Afin de spécifier clairement ces différents concepts, nous utiliserons le formalisme

2. http ://moise.sourceforge.net/

de la théorie des ensembles pour présenter les différentes dimensions d'une spécification organisationnelle.

5.2 Concept d'exigence

Le concept d'**exigence** a été défini afin d'exprimer de façon explicite des conditions ou prérequis à remplir par des agents pour leurs entrées et sorties d'organisation, pour leurs évolutions en terme d'adoption et d'abandon de rôle(s), en terme d'engagement et d'abandon de missions et buts– au sein d'une organisation. Concrètement, des exigences peuvent être définies et associées à des rôles dans une spécification structurelle, à des missions et des buts dans une spécification fonctionnelle ainsi qu'à des portails dans une spécification d'entrée / sortie (cf. figure 5.1).

L'intérêt de ce concept est lié au fait qu'une organisation a des objectifs qu'elle voudrait atteindre et qu'en général un seul agent ne peut réaliser tout seul. Ainsi, les agents qui veulent entrer dans une organisation doivent montrer « patte blanche » et déclarer leurs compétences pour leur entrée relativement à leur future participation à la réalisation des objectifs de cette organisation.

L'expression générale d'une exigence est représentée par la définition BNF 5.1.

$$
\begin{aligned}
Requirement &::= \langle type, RExpr, property \rangle \qquad (5.1)\\
type &::= EntryReq | AdoptReq | LeaveReq | ExitReq\\
RExpr &::= SRExpr \mid (RExpr\ '\wedge'\ RExpr) \mid (RExpr\ '\vee'\ RExpr)\\
SRExpr &::= (Term\ op\ Value) \mid (Term\ \ '==\ '\ \ '\_')\\
op &::= \ '==\ ' \ \mid \ '\leqslant\ ' \ \mid \ '\geqslant\ ' \ \mid \ '<\ ' \ \mid \ '>\ ' \ \mid \ '\neq\ '\\
property &::= mandatory | optional
\end{aligned}
$$

Chaque exigence est décrite par un $type$, une expression ($RExpr$) et une propriété ($property$).

1. Le **type** d'une exigence peut être $EntryReq, AdoptReq, LeaveReq, ExitReq$ qui désigne respectivement une exigence d'entrée, d'adoption, d'abandon et de sortie. Chacun de ces types est présenté plus loin dans cette section.
2. L'expression d'une exigence ($RExpr$) peut être soit une expression simple ($SRExpr$) soit une conjonction d'expressions ($RExpr \wedge RExpr$), soit une disjonction d'expressions ($RExpr \vee RExpr$).

   L'expression simple d'une exigence peut avoir l'une des deux syntaxes suivantes :

- $Term\ op\ Value$ où $Term$ représente un terme, op est l'un des opérateurs présentés par l'équation 5.1 et $Value$ est une valeur prédéfinie que doit vérifier $Term$ suivant l'opérateur op. Cette valeur dépend de l'application concernée par l'expression de l'exigence.
- $Term$ '==' '_' : cette syntaxe indique que, le terme de l'expression simple de l'exigence n'a pas de valeur prédéfinie.

3. La **propriété** d'une exigence peut être $mandatory$ ou $optional$. Lorsque la propriété *mandatory* est associée à une exigence, la valeur de vérité de son expression doit obligatoirement être vérifiée pour que l'exigence soit considérée comme ayant été satisfaite. Tandis que quand la propriété *optional* est associée à une exigence, la satisfaction de l'exigence constitue un bonus et non une obligation pour les agents. Ce bonus sera pris en compte dans le processus d'entrée, de sortie, d'adoption ou d'abandon de rôle ou de mission ou de but pour lequel l'exigence est définie.

Exemples : $e1, e2$ sont des exemples d'exigences de type $EntryReq$ spécifiant des connaissances linguistiques, l'une (e1) est obligatoire et l'autre (e2) optionnelle.

- $e1 = \langle EntryReq, LangSkill == English, mandatory \rangle$
- $e2 = \langle EntryReq, LangSkill == japanese, optional \rangle$

Dans la section ci-dessous, nous présentons les quatre types d'exigences que nous avons définis pour la gestion des entrées et des sorties ainsi que pour des engagements et les abandons d'engagements dans une organisation.

Types d'exigence

Exigence d'entrée

Une exigence d'entrée est une exigence de type $EntryReq$. Elle est définie, comme son nom l'indique, pour la gestion des entrées dans une organisation multi-agents. Elle constitue une condition que doit remplir tout agent lors de son processus d'entrée dans l'organisation dans laquelle elle est définie.

Exigence d'adoption

Une exigence d'adoption est une exigence de type $AdoptReq$. Elle doit être satisfaite par l'agent pour que les responsabilités ou tâches d'un rôle, ou d'une mission ou d'un but puissent lui être attribuées dans une organisation.

Les exigences d'entrée et d'adoption semblent similaires, mais leur sémantique respective est différente. En effet, une exigence d'entrée est définie pour des agents n'appartenant pas à un groupe ou une organisation. Elle exprime des prérequis d'entrée que doit avoir tout agent qui veut entrer dans ce groupe ou organisation. Une exigence d'adoption est définie pour exprimer un pré-requis en terme de compétence, but, capacité, ou autres conditions nécessaires pour assumer / atteindre les objectifs correspondants à un rôle, une mission ou un but. En d'autres termes, un agent devra toujours satisfaire les exigences d'adoption d'un rôle, d'une mission ou d'un but auquel il voudrait s'engager. Si l'agent n'est pas encore membre du groupe c'est-à-dire s'il n'est pas encore entré dans le groupe auquel est associé le rôle, la mission et le but, il devra également satisfaire les exigences d'entrée dans ce groupe. Dans le cas où il est déjà membre du groupe, il n'aura plus à satisfaire les exigences d'entrée.

Exigence d'abandon

Une exigence d'abandon est une exigence de type $LeaveReq$. Elle exprime une condition à vérifier lorsqu'un agent voudra se défaire des responsabilités qu'il a pour un rôle, une mission ou un but auquel il s'est engagé dans une organisation.

Exigence de sortie

Une exigence de sortie est une exigence de type $ExitReq$. Elle est définie pour un portail de gestion de sortie dans une organisation. Elle exprime une condition à vérifier lorsqu'un agent voudra quitter l'un des groupes auxquels il appartient et pour lequel les sorties d'agents sont gérées par ce portail.

Une exigence de sortie se distingue d'une exigence d'abandon dans la mesure où la première définit une condition de départ d'une organisation. Tandis que, la seconde exprime une condition d'abandon de responsabilités liées à un but, une mission, ou un rôle, mais sans que l'agent ne quitte complètement l'organisation dans laquelle il assumait cette responsabilité.

Nous présentons ci-dessous les différentes dimensions de MOISE en commençant par la dimension structurelle.

5.3 Spécification Structurelle

Définition 1 *La spécification structurelle (SS) de l'OML MOISE définit les éléments structurels d'une organisation.*

Le concept de *rôle* défini dès la version $\mathcal{M}$OISE$^+$ [81] est utilisé pour exprimer le niveau individuel de la structure d'une organisation. En d'autres termes, un rôle peut être adopté par des agents. Le concept de *groupe* a également été défini par $\mathcal{M}$OISE$^+$ pour exprimer le niveau collectif dans la SS d'une organisation. Les agents appartenant à un ou plusieurs collectif(s) au sein de la structure d'une organisation. Chaque collectif est défini par un groupe.

Ainsi, une SS est définie selon l'équation 5.2. Elle est constituée par l'ensemble de ses rôles, l'ensemble de ses groupes et une relation d'héritage entre les rôles.

$$SS = \langle R, Gr, \sqsubset \rangle \tag{5.2}$$

– R : ensemble des rôles de la SS qui comprend des rôles abstraits dont l'ensemble est noté $\mathcal{R}_{abs}$ et des rôles non abstraits que nous notons $\neg\mathcal{R}_{abs}$. Nous présentons en détail la notion de rôle dans la section 5.3.1.

$$R = \{r_i, 1 \leq i \leq n, |R| = n\},\ R = \mathcal{R}_{abs} \cup \neg\mathcal{R}_{abs}$$

– Gr : ensemble des groupes et sous groupes de la SS. Nous détaillons la notion de groupe dans la section 5.3.3.

$$Gr = \{gr_j, 1 \leq j \leq m, |Gr| = m\}$$

– $\sqsubset$: la relation d'héritage entre les rôles de R. Ainsi, lorsqu'un rôle $r \in R$ hérite d'un rôle $r' \in R$, on note $r \sqsubset r'$. Les propriétés suivantes sont définies pour cette relation :

$$\forall r, r' \in R,\ r \sqsubset r' \ \wedge\ r' \sqsubset r \implies r = r'$$
$$\forall r, r', r" \in R,\ r \sqsubset r' \ \wedge\ r' \sqsubset r" \implies r \sqsubset r"$$

Nous présentons dans les sections qui suivent les trois niveaux *individuel, social et collectif* d'une SS.

5.3.1 Niveau individuel : « Rôle »

Les rôles définis dans la spécification structurelle de l'OML MOISE permettent de représenter explicitement le niveau individuel des agents dans la structure d'une organisation.

Définition 2 *Un **rôle** est l'abstraction d'un agent dans la SS. Il sert de point d'ancrage d'un agent à un ensemble de responsabilités et contraintes qu'il doit suivre à partir du moment où il accepte de jouer ce rôle [68].*

Nous avons enrichi la spécification d'un rôle telle qu'elle était proposée dans les modèles $\mathcal{M}$OISE$^+$ et $\mathcal{M}$OISE^{Inst} avec l'intégration des notions d'exigence d'adoption et d'exigence d'abandon (cf. équation 5.3). Ainsi,

$$\forall r \in (R \setminus \mathcal{R}_{abs}),\ r = \langle AdoptReq_r, LeaveReq_r \rangle \tag{5.3}$$

- Les exigences d'adoption d'un rôle r notées $AdoptReq_r$ sont définies pour une organisation et représentent des prérequis que l'organisation demande aux agents qui voudraient adopter le rôle r.
- Les exigences d'abandon d'un rôle r notées $LeaveReq_r$ sont des conditions définies par une organisation afin d'être vérifiées lors des abandons du rôle r par un agent.

Dans une SS certains rôles peuvent être *abstraits*. Ce type de rôles ne peut être attribué à un agent dans une organisation. En général, une relation d'héritage est spécifiée entre un rôle abstrait et un ou plusieurs rôle(s) non abstrait(s). Les rôles abstraits servent alors à définir et factoriser des propriétés communes à plusieurs rôles dont les rôles non abstraits hériteront. Contrairement à un rôle abstrait, un *rôle non abstrait* est un rôle que peut jouer un agent dans une organisation.

Remarque : Nous supposons que le concepteur de la spécification structurelle s'assurera de la cohérence des exigences entre les différents rôles de façon à avoir une bonne cohérence entre des rôles qui héritent d'autres rôles.

5.3.2 Niveau social : « Lien »

Le niveau social dans une SS est exprimé au moyen des liens qui peuvent être définis entre les rôles au sein d'un groupe.

Définition 3 *Un **lien** est une relation entre deux rôles au sein d'un groupe. Il est défini par un rôle source, un rôle destination et un type [68].*

On note L_{gr} l'ensemble des liens d'un groupe gr. Un lien appartenant à L_{gr} est défini selon l'équation ci-dessous.

$$\forall l \in L_{gr},\ l = \langle r_s, r_d, t \rangle \tag{5.4}$$

– r_s est le rôle source du lien, $r_s \in R_{gr}$ où R_{gr} est l'ensemble des rôles de gr et $R_{gr} \subset R$.
– r_d est le rôle destination du lien, $r_d \in R_{gr}$.
– $t \in \{acq, com, aut\}$ est le type du lien.

Les **types de liens** entre les rôles des groupes d'une SS définis ci-dessous sont les mêmes que ceux des modèles $\mathcal{M}$OISE$^+$ [81] et $\mathcal{M}$OISE^{Inst} [68].

– Lien d'*accointance* (acq) : il spécifie que les agents jouant le rôle source r_s sont autorisés à représenter des agents jouant le rôle cible r_d.
– Lien de *communication* (com) : il exprime le fait que les agents jouant le rôle source r_s sont autorisés à communiquer avec les agents jouant le rôle cible r_d.
– Lien d'*autorité* (aut) : ce dernier type de lien permet aux agents jouant le rôle source r_s de à contrôler les agents jouant le rôle cible r_d.

La représentation graphique des liens d'une SS ainsi que de ses autres éléments est présentée sur la figure 5.2.

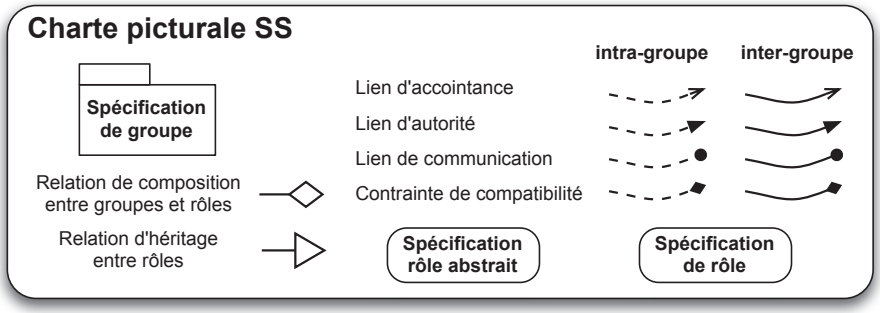


FIGURE 5.2 – Élément de représentation graphique de la SS

Remarques

1. Il existe une relation de subsumption entre les trois types de liens ci-dessus cités. Elle est définie de la façon suivante [81] :
   – $\forall l \in L_{gr},\ l = \langle r_s, r_d, aut\rangle \Rightarrow \exists l' \in L_{gr}, l' = \langle r_s, r_d, com\rangle$: s'il existe un lien d'autorité entre deux rôles r_s et r_d, alors il existe un lien de communication implicite entre ces deux rôles.
   – $\forall l \in L_{gr},\ l = \langle r_s, r_d, com\rangle \Rightarrow \exists l'' \in L_{gr},\ l'' = \langle r_s, r_d, acq\rangle$: s'il existe un lien de communication entre deux rôles r_s et r_d alors il existe un lien d'accointance implicite entre ces deux rôles.

2. Lorsqu'il existe une relation d'héritage entre des rôles, les liens définis entre rôles se propagent comme suit :

$$\forall l \in L_{gr},\ l = \langle r_s, r_d, t\rangle,\ \forall r'_s \in R_{gr}, r'_s \sqsubset r_s\ \Rightarrow\ \exists l' \in L_{gr},\ l' = \langle r'_s, r_d, t\rangle$$
$$\forall l \in L_{gr},\ l = \langle r_s, r_d, t\rangle,\ \forall r'_d \in R_{gr}, r'_d \sqsubset r_d\ \Rightarrow\ \exists l'' \in L_{gr},\ l'' = \langle r_s, r'_d, t\rangle$$

5.3.3 Niveau collectif : « Groupe »

Le niveau collectif de contrainte dans une spécification structurelle est défini au moyen des groupes de la SS. La notion de groupe a été proposée pour exprimer le cadre dans lequel des agents peuvent jouer des rôles.

Définition 4 *Un **groupe** est l'abstraction d'un collectif d'agents [68]. Il est défini par un sept-uplet présenté par l'équation 5.5.*

$$\forall gr \in Gr, gr = \langle R_{gr}, SG_{gr}, L_{gr}^{intra}, L_{gr}^{inter}, \mathcal{C}_{gr}^{intra}, \mathcal{C}_{gr}^{inter}, ng_{gr}, nr_{gr}\rangle \tag{5.5}$$

– R_{gr} : l'ensemble des rôles de gr, $R_{gr} \subset R$.

– SG_{gr} : l'ensemble des sous groupes de gr.

– L_{gr}^{intra}, L_{gr}^{inter} : respectivement l'ensemble des liens tels que définis ci-dessus, catégorisés en liens *intra-groupes* et *inter-groupes* définis pour le groupe gr. Nous reviendrons sur ces catégories de liens ci-dessous. Nous notons L_{gr} l'ensemble formé par l'union de L_{gr}^{intra} et de L_{gr}^{inter}, $L_{gr} = L_{gr}^{intra} \bigcup L_{gr}^{inter}$

– $\mathcal{C}_{gr}^{intra}, \mathcal{C}_{gr}^{inter}$: respectivement l'ensemble des contraintes de compatibilité *intra-groupes* et *inter-groupes* définies pour le groupe gr.

– ng_{gr} : la fonction qui associe à chaque sous-groupe $gr' \in SG_{gr}$ sa cardinalité en terme de nombres d'instances minimum et maximum de gr' qui peuvent être créées dans une organisation.

– nr_{gr} la fonction qui associe à chaque rôle r du groupe gr sa cardinalité en terme de le nombre d'instances minimum et maximum de r pouvant exister dans le groupe gr au sein d'une organisation.

Relation de sous-groupe

– Un groupe gr' est sous groupe d'un groupe gr si $gr' \in SG_{gr}$, on dit qu'il existe un lien de composition de gr vers gr'. La représentation graphique d'un lien de composition est donnée par la figure 5.2.

– Si un groupe n'est le sous-groupe d'aucun autre groupe alors il représente le groupe *racine* de la SS.
– Un groupe ne peut être le sous groupe que d'un seul groupe.

Dans ce qui suit, nous parlerons d'**instance de groupe** pour désigner une entité collective d'une organisation créée à partir d'une spécification de groupe définie dans la SS. Les agents peuvent entrer et adopter des rôles au sein d'une instance de groupe.

Liens intra et inter groupe(s)

On distingue deux catégories de liens entre les rôles des groupes qui précisent la *portée du lien* [81, 68] relativement au groupe gr dans lequel ils sont définis.

1. Les liens *intra-groupe* dont l'ensemble est noté L_{gr}^{intra} sont des liens dont la portée se limite à une instance de gr et aux instances de sous groupes qu'elle contient. Ainsi, $\forall l \in L_{gr}^{intra},\ l = \langle r_s, r_d, type \rangle$, tout agent qui joue le rôle r_s dans une instance de gr a un lien de type $type$ avec tous les agents qui jouent le rôle r_d dans la même instance ou dans les instances de sous groupes $gr' \in SG_{gr}$.

2. Les liens *inter-groupes* (L_{gr}^{inter}) sont des liens dont la portée concerne toute instance de gr. Ainsi, $\forall l \in L_{gr}^{inter},\ l = \langle r_s, r_d, type \rangle$, tout agent qui joue le rôle r_s dans une instance de gr a un lien de type $type$ avec tous les agents qui jouent le rôle r_d quelque soit l'instance de gr dans laquelle sont les agents qui jouent le rôle r_d.

Contraintes

Puisque plusieurs rôles peuvent être définis dans un même groupe, les notions de *contrainte de compatibilité* et de *contrainte de cardinalité* ont été proposées dans l'OML $\mathcal{M}$OISE$^+$ [81] pour contraindre des agents au niveau collectif de la SS c'est-à-dire dans des groupes définis dans une spécification structurelle.

1. Une **contrainte de compatibilité** est définie entre deux rôles et permet d'exprimer explicitement la possibilité pour un agent d'adopter les deux rôles compatibles. Elle est représentée selon l'équation 5.6.

   Une contrainte de compatibilité a également une portée qui peut être *intra-groupe* ou *inter-groupe*.
   Ainsi, on note $\mathcal{C} = \mathcal{C}_{gr}^{intra} \bigcup \mathcal{C}_{gr}^{inter}$ l'ensemble de toutes les contraintes intra et inter groupe(s) définies pour un groupe $gr \in Gr$. Un élément de $\mathcal{C}$ est défini

comme suit :

$$\forall c \in \mathcal{C}, c = r_s \bowtie r_d \tag{5.6}$$

Compatibilité intra-groupe : $\forall c \in \mathcal{C}_{gr}^{intra}$, tous les agents qui jouent le rôle source r_s dans une instance de gr peuvent également jouer le rôle r_d dans la même instance ou dans une de ses instances de sous groupes.

Compatibilité inter-groupe : $\forall c \in \mathcal{C}_{gr}^{inter}$, tous les agents qui jouent le rôle source r_s dans une instance de gr peuvent également jouer le rôle r_d dans n'importe quelle instance de gr.
La relation d'héritage entre rôles se répercute également sur les contraintes de compatibilité. Ainsi, on a :

$$r_s \bowtie r_d \ \wedge \ r_s \sqsubset r'_s \ \Rightarrow \ r'_s \bowtie r_d$$
$$r_s \bowtie r_d \ \wedge \ r_d \sqsubset r'_d \ \Rightarrow \ r_s \bowtie r'_d$$

2. **Contrainte de cardinalité :** elle permet d'exprimer les nombres minimum et maximum d'instances des sous-groupes au sein d'un groupe gr et les nombres minimum et maximum d'instances de chaque rôle d'un groupe gr. La contrainte de cardinalité sur les sous groupes est définie par la fonction ng_{gr} représentée par l'équation 5.7 . Tandis que la fonction nr_{gr} représentée par l'équation 5.8 définit la contrainte de cardinalité sur les rôles.

$$\begin{aligned} ng_{gr} : SG_{gr} &\rightarrow \mathbb{N} \times \mathbb{N} \\ gr &\mapsto (min, max) \end{aligned} \tag{5.7}$$

$$\begin{aligned} nr_{gr} : R_{gr} &\rightarrow \mathbb{N} \times \mathbb{N} \\ r &\mapsto (min, max) \end{aligned} \tag{5.8}$$

5.3.4 Exemple

L'exemple que nous utilisons dans ce chapitre pour illustrer la modélisation d'une organisation avec l'OML MOISE est appelé "write-paper". Il représente la description d'une organisation pour l'écriture collaborative d'un article par différents agents. Il est assez simple et permet ainsi de mieux présenter les différentes spécificités de chaque dimension de notre OML. L'article à rédiger concerne le domaine des organisations multi-agents.

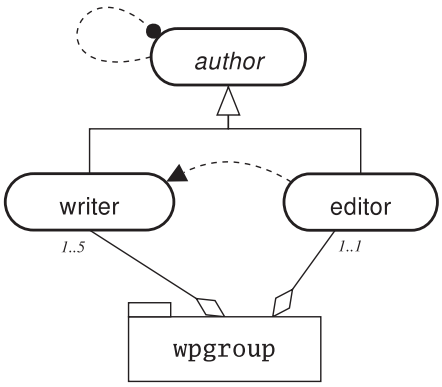


FIGURE 5.3 – Représentation graphique de la SS de l'exemple "write-paper"

La spécification structurelle de cet exemple (cf. figure 5.3), est constituée d'un groupe `wpgroup` qui comprend deux rôles *editor* et *writer* qui héritent du rôle abstrait *author*. Du fait de l'héritage, tous les agents peuvent communiquer entre eux indépendamment du (ou des) rôle(s) qu'ils jouent en raison du lien de communication défini sur le rôle *author*. Un agent jouant le rôle *editor* a autorité sur les agents qui jouent le role *writer*. Les différents éléments de la SS sont représentés ci-dessous.

$$SS = \langle R, Gr, \sqsubset \rangle$$
$$R = \{editor, writer, author\},\ R_{abs} = \{author\}$$
$$Gr = \{wpgroup\}$$
$$editor \sqsubset author,\ writer \sqsubset author$$

$$wpgroup = \langle R_{wpgroup}, SG_{wpgroup}, L^{intra}_{wpgroup}, L^{inter}_{wpgroup}, \mathcal{C}^{intra}_{wpgroup}, \mathcal{C}^{inter}_{wpgroup}, ng_{wpgroup}, nr_{wpgroup} \rangle$$

$$R_{wpgroup} = \{writer, editor\}$$
$$editor = \langle AdoptReq_{editor}, \phi \rangle$$
$$writer = \langle AdoptReq_{writer}, \phi \rangle$$
$$AdoptReq_{editor} = AdoptReq_{writer} = \{ad\_req_{01}, ad\_req_{02}\}$$
$$ad\_req_{01} = \langle AdoptReq, DomainSkill == MAS\_organisation, mandatory \rangle$$
$$ad\_req_{02} = \langle AdoptReq, LanguageSkill == English, mandatory \rangle$$
$$SG_{wpgroup} = \phi$$

$L^{intra}_{wpgroup} = \{l1, l2\}$
$l1 = \langle author, author, com \rangle$
$l2 = \langle editor, writer, aut \rangle$
$L^{inter}_{wpgroup} = \phi$
$\mathcal{C}^{intra}_{wpgroup} = \phi$
$\mathcal{C}^{inter}_{wpgroup} = \phi$
$nr_{wpgroup}(editor) = (1, 1)$
$nr_{wpgroup}(writer) = (1, 5)$

Précisions

- L'exigence d'adoption ad_req_{01} impose que le domaine de compétences des agents qui voudront adopter les rôles $editor$ et $writer$ doit être obligatoirement (cf. propriété mandatory de ad_req_{01}) relative à $MAS\_organisation$.
- L'exigence ad_req_{02} spécifie que les agents qui voudront adopter l'un ou l'autre des rôles $editor$ et $writer$ doivent obligatoirement avoir l'anglais comme compétence linguistique.
- Aucune exigence d'abandon n'est définie sur chacun de ces rôles.

5.4 Spécification Fonctionnelle

Définition 5 *La spécification fonctionnelle (FS) définit un ensemble de schémas sociaux considérés comme des objectifs collectifs à atteindre par une organisation [68].*

Plus précisément, c'est dans la FS de l'OML MOISE que l'on spécifie les objectifs collectifs d'une organisation, la décomposition de ces objectifs en *buts* qui devront être réalisés par des agents, l'ordonnancement des buts qui définit les coordinations et coopérations entre les agents au moyen de *schémas sociaux* et le regroupement des buts en *missions* auxquelles des agents pourront s'engager. Les concepts de buts, missions, et schémas sociaux permettent ainsi de définir les niveaux individuel, social, et collectif de la dimension fonctionnelle d'une organisation.

L'équation 5.9 représente la définition d'une spécification fonctionnelle qui est constituée de deux éléments. L'ensemble S des schémas sociaux et une fonction Pr qui permet de définir un ordre de préférence d'exécution des missions définies dans les schémas sociaux de S. Nous présentons sa définition dans la section 5.4.1.

$$FS = \langle S, Pr \rangle \tag{5.9}$$

5.4.1 Niveau collectif : « Schéma social »

Définition 6 *Un **schéma social** est la structure d'un objectif global d'une organisation, décomposé en buts, structurés en plans et regroupés en missions [68].*

Un schéma social permet donc de définir le processus que des agents doivent suivre pour atteindre un objectif ou but collectif. Il est représenté par un arbre appelé *plan* dont la racine est le but collectif à atteindre et dont les noeuds intermédiaires et les feuilles représentent la décomposition en sous buts du but collectif (exemple : Fig. 5.5).

La représentation formelle d'un schéma social est donnée par l'équation suivante.

$$\forall s \in S, s = \langle G_s, M_s, P_s, RespGr_s, nm_s, ng_s, mo_s \rangle \tag{5.10}$$

- G_s : l'ensemble des buts du schéma social. La notion de but est présentée dans la section 5.4.3.
- M_s : l'ensemble des missions du schéma social. La notion de mission est présentée dans la section 5.4.2.
- P_s : l'ensemble des plans du schéma social.
- $RespGr_s$: l'ensemble des groupes pouvant être responsables du schéma social.
- nm_s et ng_s : fonctions qui définissent des contraintes de cardinalités sur les missions et les buts du schéma social.
- mo_s : la fonction qui à chaque mission m de M_s associe un sous ensemble de but(s) SG_m de G_s, $SG_m \subset G_s$.

$$\begin{aligned} mo_s : M_s &\rightarrow \mathcal{P}(G_s) \\ m &\mapsto SG_m \end{aligned} \tag{5.11}$$

Expression des préférences sur les missions d'un schéma social

$$\text{Soit } M_{FS} = \bigcup_{s \in S} M_s$$

Les relations de préférences $pr \in Pr$ de la spécification fonctionnelle définissent un ordre noté $\prec$ d'exécution des missions des schémas sociaux de la FS et s'exprime comme suit.

$\forall pr \in Pr, \forall m_i, m_j \in M_{FS}, pr = m_i \prec m_j \Rightarrow m_i$ doit être réalisée avant m_j (5.12)

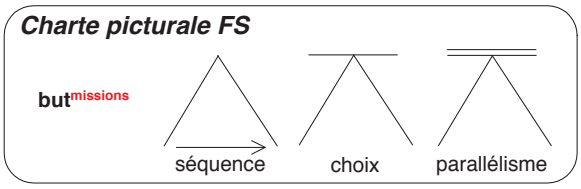


FIGURE 5.4 – Élément de représentation graphique de la FS

Plans d'un schéma social

Chaque élément de l'ensemble des plans d'un schéma social P_s est défini de la façon suivante :

$$\forall p \in P_s, p = \langle head, op, G_p \rangle \tag{5.13}$$

- $head$: est le but racine du plan, $head \in G_s$
- G_p : est l'ensemble des buts du plan.
- op : est l'opérateur du plan, $op \in \{,, |, \|\}$. La représentation graphique de ces opérateurs est présentée par la figure 5.4.
  1. L'opérateur "**,**" indique que les buts de G_p devront être exécutés *séquentiellement*.
  2. L'opérateur " | " indique qu'il faudra *choisir* parmi les buts de G_p celui qui sera exécuté. Ce choix pourra se faire selon les préférences définies dans Pr.
  3. L'opérateur " $\|$ " indique que les buts de G_p devront être exécutés en *parallèle*.

L'exemple présenté dans la section 5.4.4 illustre quelques exemples de plans.

Groupe(s) responsable(s) d'un schéma social

Pour expliciter le type de groupe de la spécification structurelle (SS) dont les agents qui jouent des rôles en son sein pourront participer à la réalisation des buts d'un schéma social, nous avons enrichi la spécification de chaque schéma social $s \in S$ avec la notion de groupe responsable d'un schéma social (*ResponsibleGroup*). Plusieurs groupes peuvent être responsables d'un même schéma social. Tout agent qui s'engagera à réaliser soit un but, soit une mission de s sera membre d'au moins un des groupes responsable de s.

Ainsi, l'ensemble $RespGr_s$ des groupes responsables d'un schéma social s est calculé par la fonction $respGroup$ définie comme suit :

$$respGroup : S \rightarrow \mathcal{P}(Gr) \tag{5.14}$$

Contraintes de cardinalité

Deux types de contraintes de cardinalité sont définis dans un schéma social. Elles sont exprimées par les fonctions nm_s et ng_s présentées ci-dessous.

1. **Contrainte de cardinalité sur les missions :** elle spécifie pour chaque mission m de M_s le nombre minimum et le nombre maximum d'agent(s) qui pourront s'engager à m dans le schéma social. Elle est définie par la fonction nm_s.

$$\begin{aligned} nm_s : M_s &\rightarrow \mathbb{N} \times \mathbb{N} \\ m &\mapsto (min, max) \end{aligned} \tag{5.15}$$

2. **Contrainte de cardinalité sur les buts :** elle définit pour chaque but g de G_s les nombres minimum et maximum d'agent(s) qui devront réaliser le but g dans le schéma social selon la fonction ng_s.

$$\begin{aligned} ng_s : G_s &\rightarrow \mathbb{N} \times \mathbb{N} \\ g &\mapsto (min, max) \end{aligned} \tag{5.16}$$

   Un but g ne sera considéré comme étant effectivement réalisé que si le nombre d'agents ayant validé sa réalisation est égal au moins à min de $ng_s(g)$.

Remarque : l'ensemble G_s des buts d'un schéma social est composé de deux sous ensembles G_s^{nodes} et G_s^{leaves}. Ainsi on a : $G_s = G_s^{nodes} \cup G_s^{leaves}$.

- G_s^{nodes} est composé des buts du schéma social qui sont des noeuds. C'est-à-dire, le but racine du schéma social ainsi que tous les autres buts qui ont des fils.
- G_s^{leaves} est composé des buts du schéma social qui sont des feuilles de l'arbre de décomposition du but racine.

Nous présentons un exemple d'ensemble G_s et de sous ensembles G_s^{nodes} et G_s^{leaves} dans la section 5.4.4.

5.4.2 Niveau social : « Mission »

Définition 7 *Une **mission** est un ensemble de buts cohérents appartenant à un même schéma social et à laquelle un agent peut s'engager [68].*

Comme nous l'avons introduit dans la section 5.1, Une mission représente le niveau social d'interprétation d'une FS dans la mesure où plusieurs agents peuvent s'engager à une mission et doivent donc coordonner leurs activités afin de réaliser au mieux la mission.

La spécification d'une mission dans MOISE a été enrichie par rapport à celle des OML $\mathcal{M}$OISE^{+} et $\mathcal{M}$OISE^{Inst} et comprend les ensembles d'exigences d'adoption et d'abandon tels que présentés par l'équation ci-dessous.

$$\forall m \in M_s, m = \langle AdoptReq_m, LeaveReq_m \rangle \tag{5.17}$$

- $AdoptReq_m$: l'ensemble des exigences d'adoption de la mission.
- $LeaveReq_m$: l'ensemble des exigences d'abandon de la mission.

5.4.3 Niveau individuel : « But »

Définition 8 *Au sein d'un schéma social, un **but** est un état final constituant un objectif vers lequel une organisation tend. Un but peut être défini par un plan décomposant le but en sous-buts [68].*

Un but est également interprété dans le modèle MOISE comme étant le niveau individuel pour la FS. En effet, les buts sont atteints ou réalisés individuellement par des agents. Cependant, selon les contraintes de cardinalité définies sur les buts d'un schéma social (Cf. fonction 5.15), certains buts peuvent nécessiter que plusieurs agents les réalisent. Dans ce cas, chaque agent concerné par la réalisation du but devra participer individuellement à sa réalisation et à la validation de la dite réalisation. L'équation 5.18 présente la spécification d'un but.

$$\forall g \in G, g = \langle AdoptReq_g, LeaveReq_g, ttf \rangle \tag{5.18}$$

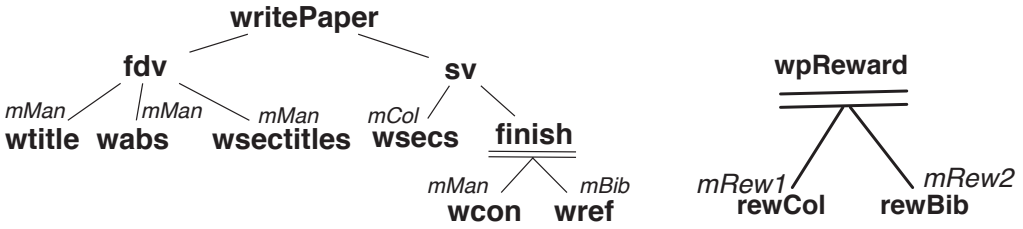
- $AdoptReq_g$ et $LeaveReq_g$ sont respectivement les ensembles des exigences d'adoption et d'abandon du but.
- ttf est la durée définie pour la réalisation du but.

Nous pouvons remarquer que comme dans la spécification d'un rôle présentée dans la section 5.3.1, la spécification d'une mission et celle d'un but permettent d'expliciter des exigences d'adoption et d'abandon : $AdoptReq_m$ et $LeaveReq_m$ pour une mission, $AdoptReq_g$ et $LeaveReq_g$ pour un but. Ces exigences sont vérifiées respectivement lorsqu'un agent veut s'engager à la réalisation d'un but ou à plusieurs buts d'une mission et lorsqu'un agent veut abandonner une mission ou un but pour lequel il s'était engagé. Nous verrons l'illustration de l'utilisation de ces exigences dans le chapitre 7.

Remarque : nous supposons que les exigences d'adoption et d'abandon définies pour les missions et les buts d'une spécification fonctionnelle sont cohérentes aux missions et buts auxquels elles sont associées. La notion de cohérence ici fait référence à une notion métier, c'est-à-dire que c'est le concepteur de l'organisation qui devra s'en assurer lors de la conception des spécifications fonctionnelle et structurelle.

5.4.4 Exemple

A la suite de la spécification structurelle de notre exemple présenté dans la section 5.3.4, nous présentons ici la FS du même exemple avec ses schémas sociaux, leurs missions et leurs buts.



(a) Schema social de "writePaper" (b) Schema social de "wpReward"

FIGURE 5.5 – Représentation graphique de la FS de l'exemple ''write-paper''

Dans cet exemple, on a un seul schéma social $wpSch$. Celui-ci est composé du but collectif à atteindre *writePaper*, qui décrit le plan de réalisation de ce but. Ce plan se décompose en deux sous plans qui doivent être réalisés séquentiellement. Le premier *fdv* consiste à définir une première version du papier à écrire. Le second *sv* consiste en la réalisation de la version du papier qui sera soumise. Le plan *fdv* se décompose en trois buts l'écriture du titre (*wtitle*), du résumé (*abs*) et des titres de section (wsectitle) du papier. Le plan *sv* est aussi composé de trois buts à exécuter séquentiellement : remplir les sections (*wsec*), la conclusion (*wcon*) et les références (*wref*). Trois missions sont définies dans ce schéma social : *mMan*, *mCol*, *mBib*. Aucune préférence sur les missions n'est spécifiée.

Nous présentons ci-dessous la description formelle de la FS de cet exemple.

$$
\begin{aligned}
&FS = \langle S, Pr \rangle \\
&\quad S = \{wpSch, wpRew\} \\
&\qquad wpSch = \langle G_{wpSch}, M_{wpSch}, P_{wpSch}, RespGr_{wpSch}, mo_{wpSch}, nm_{wpSch}, \rangle \\
&\qquad wpRew = \langle G_{wpRew}, M_{wpSch}, P_{wpRew}, RespGr_{wpRew}, mo_{wpRew}, nm_{wpRew}, \rangle
\end{aligned}
$$

- $G_{wpSch} = G^{nodes}_{wpSch} \cup G^{leaves}_{wpSch}$
  - $G^{nodes}_{wpSch} = \{writePaper, fdv, sv\}$
  - $G^{leaves}_{wpSch} = \{wtitle, wabs, wsectitles, wsecs, wcon, wref\}$
- $G_{wpRew} = G^{nodes}_{wpRew} \cup G^{leaves}_{wpRew}$
  - $G^{nodes}_{wpRew} = \{wpReward\}$
  - $G^{leaves}_{wpRew} = \{rewCol, rewBib\}$
- $M_{wpSch} = \{mMan, mCol, mBib\}$
- $M_{wpRew} = \{mRew1, mRew2\}$
- $P_{wpSch} = \{p_0, p_1, p_2, p_3\}$
  - $p_0 = \langle writePaper, |, \{fdv, sv\}\rangle$
  - $p_1 = \langle fdv, |, \{wtitle, wabs, wsectitle\}\rangle$
  - $p_2 = \langle sv, |, \{wsecs, finish\}\rangle$
  - $p_3 = \langle finish, ||, \{wcon, wref\}\rangle$
- $P_{wpRew} = \{p\}$
  - $p = \langle rewCol, ||, rewBib\rangle$
- $RespGr_{wpSch} = \{wpgroup\}$
- $RespGr_{wpRew} = \{wpgroup\}$

Les fonctions no_{wpSch} et nm_{wpSch} pour les missions de M_{wpSch}

Mission	mo_{wpSch}, mo_{wpRew}	nm_{wpSch}
mMan	$\{wtitle, wabs, wsectitles, wcon\}$	(1,1)
mCol	$\{wsecs\}$	(1,5)
mBib	$\{wref\}$	(1,1)
mRew1	$\{rewCol\}$	(1,1)
mRew2	$\{rewBib\}$	(1,1)

Spécification des missions de M_{wpSch} et de M_{wpRew}

- $mMan = \langle AdoptReq_{mMan}, LeaveReq_{mMan}\rangle$
  - $AdoptReq_{mMan} = \{ad\_req_{03}\}$
  - $LeaveReq_{mMan} = \{le\_req_{11}\}$

    $ad\_req_{03} = \langle AdoptReq, hasCapability == Management, optional\rangle$

    $le\_req_{11} = \langle LeaveReq, hasFinishedAllGoal == \_, mandatory\rangle$

- $mCol = \langle \phi, LeaveReq_{mCol} \rangle$,
  $LeaveReq_{mCol} = \{le\_req_{11}\}$
- $mBib = \langle \phi, LeaveReq_{mBib} \rangle$,
  $LeaveReq_{mBib} = \{le\_req_{11}\}$
- L'exigence d'adoption ad_req_{03} de la mission $mMan$ spécifie que l'agent qui voudra s'engager sur cette mission *peut* avoir des compétence de management. Cette exigence est optionnelle (cf. propriété $optional$).
- L'exigence d'abandon le_req_{11} est associé à chacune des mission de ce schéma social. Elle exprime l'obligation d'achever la mission avant de l'abandonner.
- $mRew1 = \langle \phi, \phi \rangle$
- $mRew2 = \langle \phi, \phi \rangle$

Spécification des buts de l'ensemble G^{leaves}_{wpSch} et G^{leaves}_{wpRew}

Aucune exigence d'engagement ni d'abandon n'a été définie pour les buts feuilles. Ainsi on a :

- $wtitle = \langle \phi, \phi, 30mn \rangle$
- $wabs = \langle \phi, \phi, 1h \rangle$
- $wsectitles = \langle \phi, \phi, 40mn \rangle$
- $wcon = \langle \phi, \phi, 1h \rangle$
- $wsecs = \langle \phi, \phi, 2day \rangle$
- $wref = \langle \phi, \phi, 1day \rangle$
- $rewCol = \langle \phi, \phi, 1day \rangle$
- $rewBib = \langle \phi, \phi, 1day \rangle$

5.5 Spécification Normative

Définition 9 *La spécification normative (NS) est l'ensemble de normes mettant en relation des rôles de la SS, des missions de la FS et un opérateur déontique*

La NS de la version courante du modèle MOISE résulte des évolutions de la définition et de la représentation d'une norme des modèles $\mathcal{M}\text{OISE}^{+}$ et $\mathcal{M}\text{OISE}^{Inst}$.

Quel que soit le modèle, la NS représente la dimension d'une organisation qui met en relation les autres dimensions d'une OS. Elle est donc définie de la façon suivante.

$$NS = \langle N \rangle$$

5.5.1 Norme

Définition 10 *Une **norme** explicite les* permissions*,* obligations *et interdictions des différents rôles d'une spécification structurelle vis-à-vis des missions d'une spécification fonctionnelle.*

Une norme définit donc les attentes de comportements attachées à un agent jouant un rôle particulier au sein d'une organisation. Nous avons vu dans le chapitre 4 les représentations respectives d'une norme dans chacun des modèles $\mathcal{M}$OISE$^+$ et $\mathcal{M}$OISE^{Inst}. Celle du modèle $\mathcal{M}$OISE$^+$ consiste en une simple mise en relation des rôles d'une SS avec des missions d'une FS par un opérateur déontique. Celle de la version $\mathcal{M}$OISE^{Inst} est plus expressive que celle de $\mathcal{M}$OISE$^+$ car elle permet de préciser, une condition d'activation de la norme, un contexte d'application de la norme, une sanction et une récompense associées à la norme, ainsi qu'un délai de validité de la norme.

La nouvelle version représentée par l'équation 5.19 dérive de la formalisation normative appelée *Normative Organisation Programming Language* (NOPL) [78] sur laquelle s'appuie notre modèle. L'objectif de ce formalisme est de simplifier la représentation d'une norme telle que proposée dans $\mathcal{M}$OISE^{Inst} de telle sorte que, l'expression d'une récompense ou d'une sanction soit elle même une norme dont l'activation dépend d'une condition et dont les autres éléments de la norme précisent entre autres le rôle le l'agent chargé de gérer la récompense ou la sanction, le délai de validité de la norme de récompense ou de sanction.

$$\forall n \in N, n \quad = \quad \langle condition, role, type, mission, ttf \rangle \tag{5.19}$$

- $condition$: comme son nom l'indique c'est une condition qui définit l'activation de la norme c'est-à-dire son changement d'état vers l'état $active$ (Cf. figure 5.6).
- $role$: désigne le rôle auquel est associée la norme.
- $type$: le type de la norme qui correspondant à l'une des opération déontique suivante : $obligation, permission, forbidden$.
- $mission$: élément de la FS représentant la mission sur laquelle porte l'opérateur déontique.
- ttf : (Time To Fulfil) est la durée au delà de laquelle une norme à l'état $active$ doit passer soit à l'état *fulfilled* soit à l'état *unfulfilled* (cf. figure 5.6).

Cycle de vie d'une norme

Le cycle de vie d'une norme n de NS est décrit par la figure 5.6. D'après la spécification d'une norme présentée dans la section ci-dessus, une norme a une condition

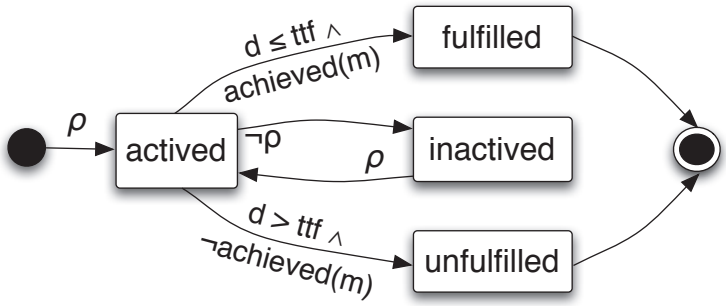


FIGURE 5.6 – Cycle de vie d'une norme

d'activation noté ρ. Lorsque celle-ci est vérifiée, la norme passe à l'état actif noté $active$. Dans cet état, l'agent ou les agents concernés par cette norme peuvent réaliser la mission (m) associée. Nous avons également vu que le délai ttf d'une norme, définit la durée pendant laquelle une norme doit être réalisée plus précisément la mission associée à la norme. Si la mission est donc réalisée au cours d'une durée d inférieure ou égale à ttf la norme passe à l'état $fulfilled$. Dans le cas contraire, la norme passe à l'état $unfulfilled$. Ces deux situations sont représentées sur la figure 5.6 respectivement par les transitions $(d \leq ttf \ \wedge \ achieved(m))$ et $(d > ttf \ \wedge \ \neg achieved(m))$. Après l'activation d'une norme, il peut arriver que la condition d'activation (ρ) change et ne soit plus valide. Dans ce cas, la norme passe à l'état $inactive$ (cf. transition $\neg\rho$ sur la figure 5.6).

Ainsi, au cours du cycle de vie d'une norme, trois éléments interviennent dans ses changements d'états : la condition d'activation, le délai de validité et la réalisation de la mission associée à la norme.

5.5.2 Exemple

Les normes définies pour notre exemple *write-paper* sont présentées dans le tableau 5.1.

- La première norme de ce tableau permet (cf. relation déontique $permission$) à tout agent qui adopte le rôle *editor* de s'engager à la mission *mMan* et a un délai de *5 jours* après l'activation de la norme pour sa réalisation.
- Les norme n_2 et n_3 obligent (cf. relation déontique $obligation$) tout agent qui adopte le rôle *writer* à s'engager aux missions *mCol* et *mBib*. Après activation de n_2, les agents qui sont engagés à *mCol* disposent d'un délai de 2 jours pour finir la mission. Après activation de n_3, les agents qui sont engagés à *mCol* ont

n_{id}	condition	role	type	mission	TTF
n_1	—	editor	$permission$	$mMan$	5 days
n_2	—	writer	$obligation$	$mCol$	2 days
n_3	—	writer	$obligation$	$mBib$	1 day
n_4	fulfilled(n_2)	editor	$obligation$	$mRew1$	1 hour
n_5	fulfilled(n_3)	editor	$obligation$	$mRew2$	1 hour

TABLE 5.1 – NS de l'exemple "write-paper"

1 jours pour finir de la réaliser.

– La norme n_4 (respectivement n_5) spécifie que, si la condition n_2 (respectivement n_3) est satisfaite, alors l'agent qui joue le rôle *editor* a l'obligation de réaliser la mission $mRew$ avec un délai de *1 heure*.

5.6 Spécification Entrée/Sortie

Définition 11 *Une spécification d'entrée / sortie (EES) définit les éléments permettant aux agents désireux d'entrer dans une organisation de savoir comment procéder pour y entrer et comment procéder pour en sortir.*

Dans cette dimension, nous nous sommes intéressés à répondre aux questions suivantes :

– Où est ce qu'un agent doit s'adresser lorsqu'il veut entrer ou sortir d'une organisation multi-agent ouverte ? Plus précisément, avec quelles entités de l'organisation doit-il interagir ?
– Quelles données sont mises à la disposition des agents pour favoriser leurs entrées / sorties au sein d'une organisation ? Plus précisément, quels sont les moyens d'entrer dans une organisation et à quelles responsabilités correspondent ces moyens ?

Pour répondre à ces besoins, nous avons défini les concepts de *portail*, *porte*, *exigence d'entrée*, *exigence de sortie* qui constituent les principaux éléments de notre spécification de procédure d'entrée / sortie dans une organisation. Ainsi une EES est composée d'un ensemble de portail (P) comme suit :

$$EES = \langle P \rangle \tag{5.20}$$

Nous présentons dans les sections qui suivent les spécifications respectives d'un *portail* et d'une *porte* et nous terminerons la présentation de l'EES avec un exemple récapitulatif (section 5.6.4).

5.6.1 Portail

Définition 12 *Un **portail** d'une organisation représente un lieu de mise à disposition et d'accès aux informations et à la gestion des procédures d'entrée et/ou de sortie.*

Nous avons défini la notion de portail d'entrée/sortie pour regrouper les fonctionnalités correspondant aussi bien à la problématique de gestion des entrées qu'à celle de gestion des sorties. Mais, lors de la définition d'une EES, il sera possible de préciser si un portail assure la double fonction de gestion des entrées et des sorties, ou s'il assure juste l'une de ces fonctions (cf. équation 5.21). Ainsi, dans une organisation on pourra avoir des portails différents pour la gestion d'une part des entrées et d'autre part des sorties. Une organisation peut avoir un ou plusieurs portails selon ses besoins ou les choix de son concepteur. Un portail est défini selon l'équation 5.21.

$$\forall p \in P, p = \langle Gr_p, Ga_p, EntryReqs_p, ExitReqs_p, type_p, hasgate_p \rangle \qquad (5.21)$$

- Gr_p : ensemble des groupes associés au portail. $Gr_p \in (\mathcal{P}(Gr) \setminus \emptyset)$ où Gr est l'ensemble des groupes de la SS (cf. équation 5.2). Dans MOISE, l'entrée dans une organisation correspond à une entrée dans un groupe, raison pour laquelle un portail est toujours associé à au moins un *groupe*, ce qui implique $Gr_p \neq \emptyset$.
- Ga_p : ensemble des portes associées au portail p. La notion de porte est présentée en détail dans la section 5.6.2
- $EntryReqs_p$: l'ensemble des exigences d'entrée définies pour le portail p.
- $ExitReqs_p$: l'ensemble des exigences de sortie définies pour le portail p.
- $type_p$: la fonction qui associe à chaque groupe appartenant à Gr_p un type de l'ensemble $\{Entry, Exit, EntryExit\}$ tel que le portail p assurera le service correspondant pour ce groupe.

$$type_p : Gr_p \longrightarrow \{Entry, Exit, EntryExit\}$$

- $hasgate_p$: la fonction qui associe à chaque groupe gr de Gr_p un sous ensemble de porte(s) de Ga qui seront les moyens d'entrer dans gr par le portail p.

$$hasgate_p : Gr_p \longrightarrow \mathcal{P}(Ga_p)$$

La représentation graphique des éléments (*portail* et *porte*) d'une EES est présentée par la figure 5.7.

FIGURE 5.7 – Représentation graphique des éléments de l'EES

5.6.2 Porte

Définition 13 *Une **porte** définit un moyen par lequel un agent peut entrer dans un groupe via un portail.*

Une porte est donc toujours définie dans un portail et est toujours associée à une cible notée $target$ qui correspond à un triplet dont le premier élément est toujours non nul et représente un *rôle* de la SS, les deux autres éléments qui peuvent être nuls sont respectivement une *mission* et un *but* de la FS. Une porte est donc définie essentiellement avec sa cible et une valeur notée dop (*duration of publication*) qui est la durée de publication de l'appel à candidature pour la porte. L'équation représentant une porte est la suivante :

$$\forall ga \in Ga_p, ga = \langle target_{ga}, dop_{ga} \rangle \tag{5.22}$$

– $target_{ga}$ est la cible de la porte. La valeur d'une cible peut avoir l'une des formes suivantes :

  – $(r, \_, \_)$: la cible est dans ce cas constituée d'un seul élément qui est un rôle défini dans l'ensemble des rôles non abstraits de la SS ($R \setminus R_{abs}$).

  – $(r, m, \_)$: la cible comprend deux éléments : un rôle non abstrait de la SS et une mission de la FS. La mission m doit être associée au rôle r par une norme de la NS.

  – (r, m, g) : la cible est dans ce cas constituée de trois éléments : un rôle non abstrait de la SS, une mission de la FS et un des buts de la mission ce but doit être un but feuille. De même que pour le cas précédent, la mission m doit être associée au rôle r par une norme de la NS.

$$target_{ga} \in (R \setminus R_{abs}) \text{ x } (M_{FS} \cup \{null\}) \text{ x } (G^{FS}_{leaves} \cup \{null\})$$

Les éléments des ensembles M_{FS} et G^{FS}_{leaves} sont respectivement les missions de tous les schémas sociaux de la FS et les buts feuilles des schémas sociaux de la

FS.

$$M_{FS} = \bigcup_{s}^{FS} M_s \qquad G_{FS}^{leaves} = \bigcup_{s}^{FS} G_s^{leaves}$$

– dop_{ga} est la durée de publication de l'appel à candidatures d'agents pour la porte. Au cours de la durée dop_{ga}, des agents pourront soumettre des candidatures pour le besoin représenté par la porte ga.

Remarque : appartenance d'un agent à un groupe

Un agent est membre d'une instance de groupe signifie qu'il a effectué une procédure d'entrée dans cette instance avec succès. La seule exception à cette règle est lorsqu'un agent entre dans une instance de groupe qui a un super groupe c'est-à-dire lorsqu'on a une hiérarchie de groupes avec des groupes qui ont des sous-groupes. Dans ce cas, lorsqu'un agent est admis dans une instance de sous-groupe, il est également membre du ou des instances de groupe(s) parent(s) de cette instance de sous-groupe. Cependant, l'inverse n'est pas vrai. En effet, lorsqu'un agent entre dans une instance de groupe qui a des instances de sous-groupes, cet agent n'est pas automatiquement membre de ces instances de sous-groupes. Il n'en sera membre que s'il y est admis à l'issue de l'exécution d'un processus explicite d'entrée.

Nous reviendrons sur cette notion d'appartenance d'un agent à un groupe dans le prochain chapitre.

5.6.3 Relations avec les autres dimensions

Les relations entre EES et les autres dimensions d'une spécification organisationnelle que nous décrivons ci dessous sont représentées sur la figure 5.1 du méta-modèle de l'OML MOISE.

Relation EES – SS

La relation entre une EES et une SS est établie d'une part avec les groupes de la SS pour lesquels les portails gèrent les entrées / sorties des agents et d'autre part avec les rôles qui peuvent être des portes d'entrée des portails de l'EES.

Relation EES – FS

Une EES est liée à une FS dans la mesure où des missions ou des buts de la FS

peuvent être définis comme portes dans la EES. Cependant, pour éviter des conflits ou ambiguïtés pouvant être engendrés par les relations –relations définies par des normes de la NS– qui existent entre des rôles de la SS et des missions de la FS, nous avons défini les contraintes de spécification de procédure d'entrée / sortie.

Contraintes de spécification de portes

Dans une spécification d'entrée / sortie, on peut avoir plusieurs *portes*, chacune étant associée à un triplet constitué d'un rôle, d'une éventuelle mission et d'un éventuel but. Compte tenu du fait qu'une mission est composée de but(s) et qu'un rôle peut être lié à une ou plusieurs missions (cf. figure 5.1) il est possible d'avoir des doublons de portes dans l'ES. Afin d'éviter ces situations, nous avons défini trois contraintes présentées ci-dessous. Ces contraintes doivent être satisfaites pour que la spécification EES soient considérée comme ayant été bien définies.

Considérons une spécification organisationnelle (OS) définie comme suit :

$$OS = \langle SS, FS, NS, EES \rangle \tag{5.23}$$

Nous définissons les fonctions suivantes portant sur les éléments d'une OS.

- $roleN(n)$: retourne le rôle de la norme n.
- $missionN(n)$: renvoie la mission associée à la norme n.
- $missionR(r)$: retourne toutes les missions associées à un rôle r par des normes $n \in NS$.

$$missionR : R \longrightarrow \mathcal{P}(M)$$
$$r \longmapsto \{m \in \mathcal{P}(M),\ \exists n \in N, roleN(n) = r \ \wedge \ missionN(n) = m\}$$

- $roleM(m)$: retourne tous les rôles associés à une mission m par des normes $n \in NS$.

$$roleM : M \longrightarrow \mathcal{P}(R)$$
$$m \longmapsto \{r \in \mathcal{P}(R),\ \exists n \in N, missionN(n) = m \ \wedge \ roleN(n) = r\}$$

- $goalM(m)$: retourne l'ensemble des buts d'une mission.

$$goalM : M \longrightarrow \mathcal{P}(G)$$
$$m \longmapsto \{g \in \mathcal{P}(G), g \in G_m\}$$

- $missionG(g)$: retourne toutes les missions attachées un but

$$missionG : G \longrightarrow M$$
$$g \longmapsto \{m \in M,\ g \in mo(m)\}$$

– $groupGa_p(ga)$: retourne le groupe associé à une porte ga d'un portail p.

$$\begin{aligned} groupGa_p \ : Ga_p &\longrightarrow Gr_p \\ ga &\longmapsto gr \end{aligned}$$

1. Lorsque la cible d'une porte ga d'un portail p, associée à un groupe gr est de la forme $(r, \_, \_)$, il ne peut y avoir d'autres portes de n'importe quel portail associées à ce groupe gr dont la cible soit de la forme $(r, m, \_)$ ou (r, m, g).
   m représente toute mission associée à r par une norme de la NS, et g représente tout but d'une mission m.

$$\begin{aligned} &\forall ga \in Ga_p,\ \exists gr \in Gr,\ \exists r \in R, \\ &\quad groupGa_p(ga) = gr \ \wedge\ target_{ga} = (r, \_, \_) \\ &\quad\Longrightarrow\quad \forall m \in missionR(r),\ \forall g \in goalM(m),\ \forall p', p'' \in P, \\ &\qquad\qquad \big(\nexists ga_1 \in Ga_{p'},\ groupGa_{p'}(ga_1) = gr \ \wedge\ target_{ga_1} = (r, m, \_)\big) \\ &\qquad\quad \wedge\ \big(\nexists ga_2 \in Ga_{p''},\ groupGa_{p''}(ga_2) = gr \ \wedge\ target_{ga_2} = (r, m, g)\big) \end{aligned}$$

2. Lorsque la cible d'une porte ga d'un portail p, associée à un groupe gr est de la forme $(r, m, \_)$, il ne peut y avoir d'autres portes de n'importe quel portail associé à ce groupe gr dont la cible soit de la forme $(r, \_, \_)$ ou (r, m, g). g représente tout but de la mission m.

$$\begin{aligned} &\forall ga \in Ga_p,\ \exists gr \in Gr,\ \exists r \in R,\ \exists m \in M_{FS}, \\ &\quad groupGa_p(ga) = gr \ \wedge\ target_{ga} = (r, m, \_) \qquad (5.24) \\ &\quad\Longrightarrow\quad \forall g \in goalM(m),\ \forall p', p'' \in P, \\ &\qquad\qquad \big(\nexists ga_1 \in Ga_{p'},\ groupGa_{p'}(ga_1) = gr \ \wedge\ target_{ga_1} \neq (r, \_, \_)\big) \\ &\qquad\quad \wedge\ \big(\nexists ga_2 \in Ga_{p''},\ groupGa_{p''}(ga_2) = gr \ \wedge\ target_{ga_2} \neq (r, m, g)\big) \end{aligned}$$

3. Lorsque la cible d'une porte ga d'un portail p, associée à un groupe gr est de la forme (r, m, g), il ne peut y avoir d'autres portes de n'importe quel portail associées à ce groupe gr dont la cible soit de la forme $(r, m, \_)$ ou $(r, \_, \_)$.

$$\begin{aligned} &\forall ga \in Ga_p,\ \exists gr \in Gr,\ \exists r \in R,\ \exists m \in M_{FS},\ \exists g \in G^{FS}_{leaves},\ g \in mo(m), \\ &\quad groupGa_p(ga) = gr \ \wedge\ target_{ga} = (r, m, g) \qquad (5.25) \\ &\quad\Longrightarrow\quad \forall p', p'' \in P, \\ &\qquad\qquad \big(\nexists ga_1 \in Ga_{p'},\ groupGa_{p'}(ga_1) = gr \ \wedge\ target_{ga_1} \neq (r, m, \_)\big) \\ &\qquad\quad \wedge\ \nexists\big(ga_2 \in Ga_{p''},\ groupGa_{p'}(ga_2) = gr \ \wedge\ target_{ga_2} \neq (r, \_, \_)\big) \end{aligned}$$

Relation EES – NS

La relation entre une EES et une NS est indirecte. En effet, les normes d'une NS définissent des liens directs entre une SS et une FS. Dans l'EES, les normes de la NS permettent d'une part de vérifier la cohérence des rôles associés aux différentes cibles ($target$) des portes d'entrées définis dans les portails de l'EES, en particulier lorsqu'une cible est une mission ou un but. D'autre part, les normes de la NS permettent aussi de vérifier que les contraintes de spécification de portes d'entrée ci-dessus présentées sont respectées.

Enfin, les normes de la NS sont aussi utilisées dans la gestion proprement dite des processus des entrées et des sorties d'agents dont les procédures sont définies dans l'EES. Concrètement nous verrons dans le prochain chapitre qu'à l'issue de l'exécution d'un processus d'entrée, un *contrat* est établi pour chaque agent admis. Chaque contrat comprend des *clauses* qui correspondent à des normes définies dans la NS.

5.6.4 Exemple

L'ensemble des portails P de la spécification d'entrée / sortie de notre exemple est composée d'un seul portail : $EES = \langle P \rangle$, $P = \{wpPor\}$. le portail $wpPor$ est associé au groupe $wpgroup$ et comprend deux portes ga_1, ga_2. EES est sommairement représentée par la figure 5.8

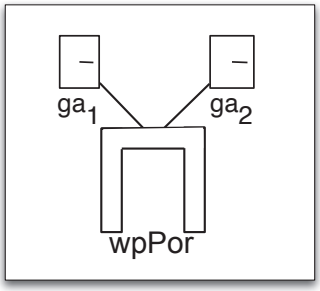


FIGURE 5.8 – Éléments de l'EES de l'exemple ''write-paper''

Spécification du portail $wpPor$ et des portes ga_1, ga_2

- $wpPor = \langle Grp_{wpPor}, Ga_{wpPor}, EntryReqs_{wpPor}, ExitReqs_{wpPor}, type_{wpPor}, hasgate_{wpPor} \rangle$
  - $Grp_{wpPor} = \{wpgroup\}$

  - $Ga_{wpPor} = \{ga_1, ga_2\}$
  - $hasgate_{wpPor}(wpgroup) = \{ga_1, ga_2\}$
  - $EntryReqs_{wpPor} = \{en\_req_1\}$
    $en\_req_1 = \langle EntryReq, hasSkill == MAS, mandatory\rangle$
  - $ExitReqs_{wpPor} = \phi$
  - $type_{wpPor}(wpgroup) = EntryExit$
- $ga_1 = \langle target_{ga_1}, 5\ day\rangle, \quad target_{ga_1} = (writer, \_, \_)$
- $ga_2 = \langle target_{ga_2}, 6\ day\rangle, \quad target_{ga_2} = (editor, mMan, \_)$

5.7 Synthèse

Nous avons présenté dans ce chapitre, les différentes dimensions de la nouvelle version du modèle MOISE, en nous focalisant sur les éléments essentiels de cet OML relatifs à l'ouverture pour la modélisation des organisations. Nous avons ainsi vu le concept d'exigence et ses types (AdoptRequirement et LeaveRequirement) qui enrichissent les spécifications structurelles et fonctionnelles, ainsi que les concepts de la nouvelle dimensions d'entrée / sortie à savoir *portal, EntryRequirement, ExitRequirement* et *gate*.

L'utilisation de portails (*portals*) offre divers avantages à une organisation parmi lesquels :

- La gestion explicite des entrées et des sorties dans une organisation par des entités dédiées.
- La possibilité de mutualiser la gestion des entrées et des sorties dans différents groupes d'une organisation en délégant la gestion des entrées et / ou des sortie dans ces groupes à un même portail.
- La possibilité de distribuer la gestion des entrées et des sorties en ayant plusieurs portails dans une organisation.

L'utilisation de portes (*gates*) permet à une organisation d'offrir plusieurs moyens d'entrer aux agents. Grâce à la richesse des versions précédentes de MOISE, nous avons désormais trois moyens d'entrer dans une organisation : *rôle*, *mission* ou *but*. Cette diversité offre une flexibilité de procédures d'entrées très intéressante pour l'ouverture d'organisations multi-agents. De plus, elle permet de pouvoir expliciter les attentes d'une organisation vis-à-vis d'agents qui entrent dès le processus d'entrée. Par exemple, un agent qui entre dans une organisation au moyen d'un *but*, s'engage

directement à la réalisation de ce *but*.

Certains éléments dont hérite l'OML MOISE des versions précédentes n'ont pas été présentés. Nous invitons les lecteurs intéressés à découvrir toute la richesse des modèles $\mathcal{M}$OISE^{+} [81, 77, 84] et $\mathcal{M}$OISE^{Inst} [70, 68, 69] en se reportant aux références ci-dessus citées.

Chapitre 6

Gestion d'entité organisationnelle ouverte

Nous avons présenté dans le chapitre précédent le langage de modélisation d'organisation MOISE. Il est utilisé pour définir les dimensions structurelle, fonctionnelle, normative et d'entrée / sortie qui constituent la spécification organisationnelle (OS) d'une organisation multi-agent. Une OS contribue à l'ouverture des organisations dans la mesure où elle représente l'une des données échangées entre une entité organisationnelle et des agents extérieurs. Ces derniers l'utiliseront pour raisonner sur l'entité organisationnelle et décider de leur entrée et de leur participation à la réalisation des objectifs collectifs de l'entité organisationnelle. Par ailleurs, les entités organisationnelles sont également créées et gérées dans une infrastructure de gestion d'organisations à partir d'une spécification organisationnelle.

L'objectif d'une entité organisationnelle étant de gérer la dynamique structurelle et fonctionnelle d'une organisation, dans ce chapitre nous présentons les principaux éléments qui participent à cette dynamique. La section 6.1 est donc consacrée à la présentation des instances de groupes, de schéma sociaux et de portails. Ensuite, (cf. section 6.1.6) nous présentons la structure d'un contrat, élément qui établit l'entrée d'un agent dans une entité organisationnelle et qui permet de gérer sa participation dans l'entité organisationnelle. Nous présentons dans la section 6.2 notre approche de gestion des entrées / sorties. Nous terminons ce chapitre par la présentation de notre approche de contrôle ou régulation des agents au sein d'une organisation.

6.1 Entité organisationnelle

Dans le chapitre 3, nous avons donné la définition d'une *entité organisationnelle* (OE) comme étant *une instance d'organisation constituée d'une spécification organisationnelle (OS), d'agents et de tout ce qui contribue à la gestion de la dynamique : évolution ou changement d'état de l'instance de l'organisation* [20, 68].

Une entité organisationnelle est créée dans un système multi-agent (SMA). Ce dernier est lui même constitué de divers éléments parmi lesquels nous considèrerons essentiellement l'ensemble des agents ($\mathcal{A}$) et l'ensemble des entités organisationnelles (Org). Ainsi, nous utiliserons dans nos travaux la représentation simplifiée suivante d'un SMA :

$$SMA = \langle \mathcal{A}, Org \rangle \tag{6.1}$$

Chaque élément de Org est une entité organisationnelle décrite à un instant t selon l'équation 6.2. Un agent ag de $\mathcal{A}$ deviendra ***membre*** d'un élément oe de Org après son admission via un processus d'entrée que nous présentons dans la section 6.2.1.

$$\begin{aligned} \forall oe \in Org,\, oe = \langle\, & OS,\, IG,\, IS,\, IP,\, Ca,\, Co,\, grpType,\, schType,\, porType, \\ & isubGrp, igrpIS, ischIG, iporIG, iporCa, caIgrp, igrpCo, \\ & cfacStatus,\, missionStatus,\, goalStatus,\, normStatus \,\rangle^t \end{aligned} \tag{6.2}$$

- OS : spécification organisationnelle de l'oe, $OS = \langle SS, FS, NS, EES \rangle$.
- IG : ensemble d'identifiants des instances de groupes créés au sein de oe.
- IS : ensemble d'identifiants des instances de schémas sociaux créés au sein de oe.
- IP : ensemble d'identifiants des instances de portails créés au sein de oe.
- Ca : ensemble d'identifiants d'appels à candidatures de l'oe. Nous présentons la notion d'appel à candidature dans la section 6.1.5.
- Co : ensemble des identifiants des contrats d'agents de l'oe. Nous présentons la notion de contrat dans la section 6.1.6.
- $grpType$: fonction qui associe un type de groupe noté gr, défini dans l'ensemble Gr des types de groupes de la spécification structurelle (SS) de l'OS, à une instance de groupe dont l'identifiant noté igr est un élément de IG.

$$\begin{aligned} grpType : IG &\longrightarrow Gr \\ igr &\longmapsto gr \end{aligned}$$

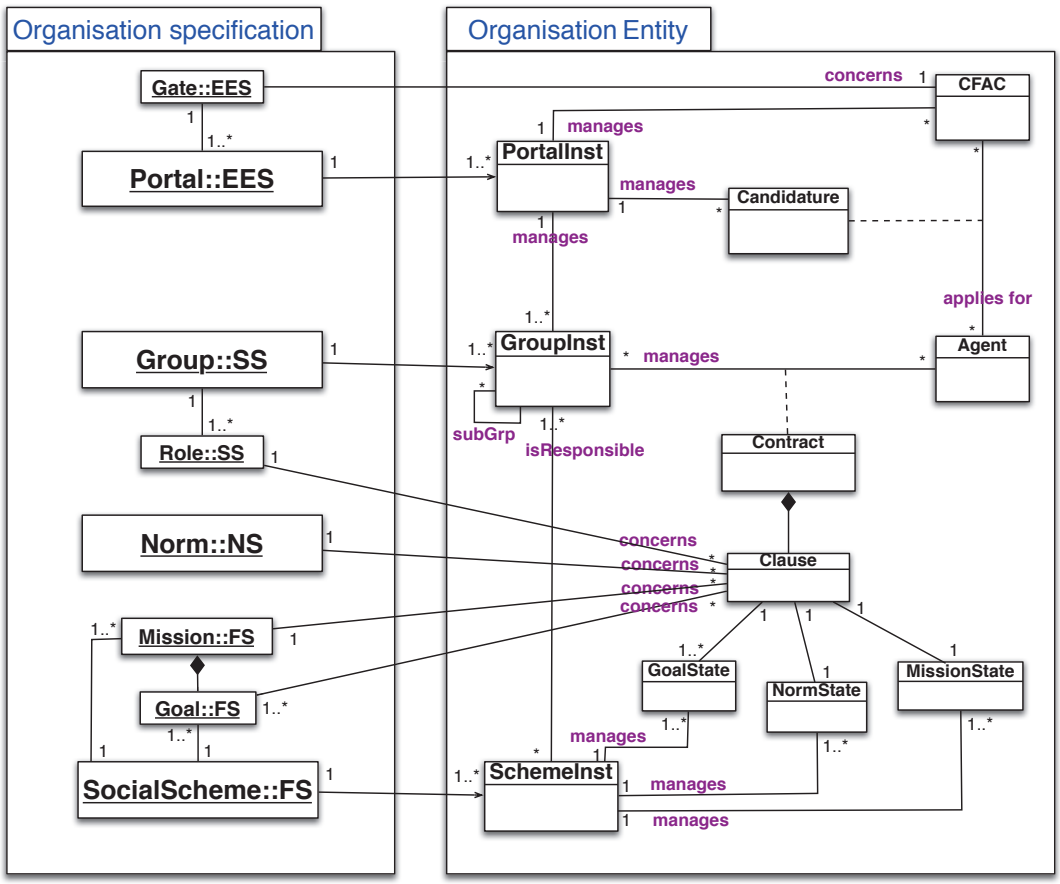


FIGURE 6.1 – Composants d'une entité organisationnelle

– $schType$: fonction qui associe un type de schéma social noté s défini dans l'ensemble des types de schémas sociaux S de la spécification fonctionnelle (FS) à une instance de schéma social dont l'identifiant $isch$ est élément de IS.

$$schType : IS \longrightarrow S$$
$$isch \longmapsto s$$

– $porType$: fonction qui associe un type de portail noté p défini dans l'ensemble des portails P de la spécification d'entrée sortie EES à une instance de portail dont l'identifiant noté $ipor$ est élément de IP.

$$porType : IP \longrightarrow P$$
$$ipor \longmapsto p$$

– $isubGrp$: fonction qui fournit les instances de sous-groupes d'une instance de groupe. Une instance de groupe igr' peut être sous-groupe d'une autre instance de groupe igr si et seulement si le type de l'instance de igr' est défini dans l'ensemble SG_{gr} des types de sous-groupes de gr. Ce dernier étant le type de igr.

$$\begin{aligned} isubGrp : IG &\longrightarrow \mathcal{P}(IG) \\ igr &\longmapsto \big\{igr',\, igr' \in IG \wedge \big(\, grpType(igr) = gr, \\ &\qquad grpType(igr') = gr',\ gr' \in SG_{gr}\,\big)\big\} \end{aligned}$$

– $ischIG$: fonction qui fournit les instances de groupes qui sont responsables d'une instance de schéma social. Une instance de groupe igr est responsable d'une instance de schéma social $isch$ si et seulement si le type de igr noté gr appartient à l'ensemble $RespGr_s$ des types de groupes pouvant être responsables de s. Ce dernier étant le type de $isch$.

$$\begin{aligned} ischIG : IS &\longrightarrow \mathcal{P}(IG) \\ isch &\longmapsto \big\{igr,\, igr \in IG \wedge \big(\, grpType(igr) = gr, \\ &\qquad schType(isch) = s,\ gr \in RespGr_s\,\big)\big\} \end{aligned}$$

– $igrpIS$: fonction qui fournit les instances de schémas sociaux dont est responsable une instance de groupe. Cette fonction est en quelque sorte l'inverse de la fonction $ischIG$.

$$\begin{aligned} igrpIS : IG &\longrightarrow \mathcal{P}(IS) \\ igr &\longmapsto \big\{isch,\, isch \in IS \wedge \big(\, grpType(igr) = gr, \\ &\qquad schType(isch) = s,\ gr \in RespGr_s\,\big)\big\} \end{aligned}$$

– $iporIG$: fonction qui fournit les instances de groupes dont une instance de portail gère les entrées / sorties. Une instance de portail $ipor$ peut gérer les entrées / sorties d'une instance de groupe igr si et seulement si le type de igr noté gr appartient à l'ensemble Gr_p des types de groupe dont p peut gérer les entrées / sortie. p est le type de l'instance de portail $ipor$.

$$\begin{aligned} iporIG : IP &\longrightarrow \mathcal{P}(IG) \\ ipor &\longmapsto \big\{igr,\, igr \in IG \wedge \big(\, grpType(igr) = gr, \\ &\qquad porType(ipor) = p, gr \in Gr_p\,\big)\big\} \end{aligned}$$

– $iporCa$: fonction qui fournit les appels à candidatures gérés par une instance de portail. Un appel à candidature noté ca est toujours créé dans une instance de portail $ipor$ et pour une porte définie dans l'ensemble Ga_p des portes associé au type de portail p. Ce dernier étant le type de portail de $ipor$.
La porte d'un appel à candidature noté ga est obtenue par la fonction $gateCa(ca)$ définie dans la section 6.1.5.

$$\begin{aligned} iporCa : IP &\longrightarrow \mathcal{P}(Ca) \\ ipor &\longmapsto \{ca,\, ca \in Ca \wedge \big(\, gateCa(ca) = ga, \\ &\qquad porType(ipor) = p,\ ga \in Ga_p \,\big)\} \end{aligned}$$

– $caIgrp$: fonction qui fournit l'instance de groupe à laquelle est associé un appel à candidature. Un appel à candidature ca est associé à une instance de groupe igr si et seulement si le type du groupe de igr noté gr appartient à l'ensemble Gr_p, p étant le type de portail de $ipor$ et si la porte associée à ca appartient au résultat de la fonction $hasgate_p(gr)$. Cette dernière fournit toutes les portes de p associé au type de groupe gr.

$$\begin{aligned} caIgrp\ &: Ca \longrightarrow IG \\ &\quad ca \longmapsto igr \\ igr \in IG\ &: \exists ipor \in IP \wedge \Big(igr \in iporIG(ipor) \wedge \\ &\quad \big(\exists ca \in caPor(ipor),\, \big(\, gateCa(ca) = ga, \\ &\qquad porType(ipor) = p,\, grpType(igr) = gr, ga \in hasgate_p(gr)\big)\big)\Big) \end{aligned}$$

– $igrpCo$: fonction qui fournit les contrats gérés par une instance de groupe. Nous verrons dans la section 6.1.6 qu'un contrat est toujours créé suite à un processus d'entrée réussi d'un agent dans une instance de groupe. Ainsi chaque contrat est associé à une seule instance de groupe et une instance de groupe peut gérer plusieurs contrats cf. Figure 6.1.

$$\begin{aligned} igrpCo : IG &\longrightarrow \mathcal{P}(Co) \\ igr &\longmapsto \{co,\, co \in Co\} \end{aligned}$$

– $cfacStatus$: fonction qui retourne l'état d'un appel à candidature. Un appel à candidature peut avoir l'un des deux états suivants : $published$ qui est l'état attribué lorsque l'appel à candidature est créé et $closed$ qui est l'état attribué lorsque le délai correspondant à la durée de publication est expiré. Ainsi après

la création d'un appel à candidature, celui-ci reste à l'état $published$ jusqu'à ce que la durée de publication soit expirée.

$$\begin{aligned} cfacStatus : Ca &\longrightarrow CfacState \\ ca &\longmapsto cfacState \\ CfacState &= \{published, closed\} \end{aligned}$$

– $missionStatus$: fonction qui fournit l'état d'une mission d'une instance de schéma social relative à une clause d'un contrat. Nous verrons dans la section 6.1.6 qu'un contrat est composé d'un ensemble de *clauses*. Chaque clause d'un contrat précise une mission à laquelle doit s'engager l'agent ainsi que l'instance de schéma social qui gère la mission. Si l'agent ne s'est pas encore engagé à une mission m d'une clause de son contrat co dans l'instance $isch$ de schéma social correspondant, la fonction $missionStatus$ retourne l'état $uncommitted$. Dans le cas contraire elle retourne l'état $committed$.

$$\begin{aligned} missionStatus : IS \text{ x } Co \text{ x } M_S &\longrightarrow MissionState \\ (isch, co, m) &\longmapsto mState \\ MissionState &= \{uncommitted, committed\} \end{aligned}$$

– $goalStatus$: fonction qui retourne l'état d'un but d'une instance de schéma social. Nous verrons dans la section 6.1.3 qu'une instance de schéma social gère essentiellement des missions et des buts. La gestion des buts consiste principalement à la coordination des conditions liées à leurs réalisations. Ainsi, lorsque les conditions nécessaires à la réalisation d'un but ne sont pas remplies, le but est à l'état $waiting$, dans le cas contraire il est dans l'état $active$. Un but passe de l'état $active$ à $achieved$ lorsqu'il a été réalisé ou à l'état $impossible$ lorsqu'il est irréalisable.

$$\begin{aligned} goalStatus : IS \text{ x } G_S &\longrightarrow GoalState \\ (isch, g) &\longmapsto gState \\ GoalState &= \{waitting, actived, achieved, impossible\} \end{aligned}$$

– $normStatus$: fonction qui retourne l'état d'une norme associée à une clause d'un contrat. Nous verrons dans la section 6.1.6 que chaque clause d'un contrat est spécifiée relativement à une norme n définie dans l'ensemble N des normes

de la spécification normative. Les valeurs pouvant être retournées par la fonction $normStatus$ sont celles du cycle de vie d'une norme Cf. Figure 5.6.

$$\begin{aligned} normStatus :\ & Co \times N \longrightarrow NormState \\ & (co, n) \longmapsto nState \\ & NormState = \{waitting, actived, inactived, fulfilled, unfulfilled\} \end{aligned}$$

Nous présentons dans les sections ci-dessous les principales fonctions d'une instance de groupe, de schéma social et de portail. Ensuite nous présentons les notions d'appel à candidature et de contrat.

6.1.1 Agents d'une entité organisationnelle

Dans une instance d'entité organisationnelle $oe \in Org$, nous distinguons deux types d'agents : les *agents de domaines* et des *agents organisationnels*.

1. Les **agents de domaines** sont ceux qui quand ils sont admis dans une OE à l'issus d'un processus d'entrée, vont s'engager aux missions des schémas sociaux et coopérer à la réalisation des objectifs collectifs de l'entité organisationnelle.

2. Les **agents organisationnels** *OrgAgent*, ont des responsabilités complémentaires aux fonctionnalités des instances de groupes, schémas sociaux et portails présentés ci-dessus.

   En effet, notre approche de gestion d'organisations ouvertes avec l'OML MOISE et l'OMI ORA4MAS que nous présentons dans le prochain chapitre, est basée sur une séparation de la gestions d'actions organisationnelles et des aspects décisionnels au sein d'une entité organisationnelle. Ainsi, les instances de portail, de groupe et de schéma social gèrent des actions organisationnelles et chaque instance est supervisée par un agent organisationnel qui contrôle les résultats d'exécution des actions organisationnelles par les agents de domaines. Cette approche a pour avantage de réduire la complexité des codes des agents dits *manager* (OrgManager, et agents d'arbitrages) d'une OE et de favoriser la mise en oeuvre des propriétés d'ouverture notamment les mécanismes d'interopérabilités, de gestion des entrées et intégrations des agents, de gestion de leur coopération, du contrôle de leurs comportements et des sorties.

Dans une entité organisationnelle, les *OrgAgent* sont donc chargés de prendre des décisions liées à la gestion du contrôle des comportements des agents de domaines relativement aux normes de l'entité organisationnelle.

6.1.2 Instance de groupe

Chaque instance de groupe d'une entité organisationnelle gère des rôles de sa spécification qui sont joués par des agents. En effet, une instance de groupe est créée à partir d'un type de groupe défini dans la spécification structurelle. D'après l'équation 5.5, un type de groupe comprend entre autre un ensemble de rôles. Ces derniers peuvent être adoptés par des agents. Ainsi, l'entrée d'un agent dans une instance de groupe se fait par l'adoption d'un rôle. Les adoptions de rôles d'une instance de groupe sont gérées par l'instance de portail qui gère les entrées des agents dans cette instance de groupe. Nous présentons le processus d'entrée dans la section 6.2.1. Lorsqu'un agent entre dans une instance de groupe, un contrat pour cet agent est créé et enregistré dans cette instance de groupe. Chaque agent d'une instance de groupe a un seul contrat au sein de cette instance. Ce contrat comporte toutes les clauses relatives à chaque rôle adopté dans l'instance de groupe.

Une instance de groupe peut également avoir des instances de sous-groupes (Cf. fonction $isubGrp$ définie dans l'équation 6.2 présentée ci-dessus). Une instance de groupe qui a des instances de sous-groupes gère la vérification des cardinalités d'instances de sous-groupes spécifiées par la fonction ng_{gr} de l'équation 5.5.

Par ailleurs, dans une entité organisationnelle, une instance de groupe est généralement responsable d'une ou plusieurs instances de schémas-sociaux (Cf. fonction $igrpIS$ définie dans l'équation 6.2 et Figure 6.1). Les clauses des contrats des agents d'une instance de groupe sont relatives aux missions gérées dans les instances de schémas-sociaux dont est responsable l'instance de groupe.

La gestion des agents jouant des rôles dans une instance de groupe se fait donc au moyen des clauses de leur contrat respectif, des *cardinalités des rôles* et des *contraintes de compatibilité intra et inter groupes* ainsi que les *liens intra et inter groupes* entre rôles qui sont définis dans sa spécification (Cf. Équation 5.5) et qui régissent la structure de l'instance de groupe. Rappelons que les contraintes de compatibilité et les liens intra-groupes ont une portée qui inclue l'instance de groupe à ses instances de sous-groupes (Cf. section 5.3.3). Cette gestion intègre également la gestion des abandons de rôles et donc des exigences d'abandons associées aux rôles Cf. 5.3. Nous reviendrons sur cette gestion dans la section 6.2.2.

En résumé, étant donnée une instance de groupe d'une entité organisationnelle, on détermine :

- Son type à partir de la fonction $grpType$ de l'équation 6.2 et donc la spécification du groupe définie dans la SS de l'entité organisationnelle.
- Ses instances de sous-groupes à partir de la fonction $subGrp$.

– Les instances de schémas sociaux dont elle est responsable à partir de la fonction $igrpIS$.
– Les agents de l'instance de groupe à partir des contrats obtenus par la fonction $igrpCo$.

Appartenance d'un agent à un groupe

Un agent est *membre* d'une instance de groupe signifie qu'il a réussi la procédure d'entrée dans cette instance et donc qu'il a un contrat enregistré dans cette instance de groupe. La seule exception à cette règle est lorsqu'un agent entre dans une instance de groupe qui est sous-groupe d'une autre instance de groupe. Dans ce cas, lorsqu'un agent est admis dans une instance de sous-groupe, il est également membre du ou des instances de groupe(s) parent(s) de cette instance de sous-groupe. Cependant, l'inverse n'est pas vrai. En effet, lorsqu'un agent entre dans une instance de groupe qui a des instances de sous-groupes, cet agent n'est pas automatiquement membre de ces instances de sous-groupes. Il n'en sera membre que s'il y est admis à l'issue de l'exécution d'un processus explicite d'entrée.

La formalisation de cette notion d'appartenance d'un agent à une instance de groupe est représentée par la fonction *ismember(ag, igr)* présentée ci-dessous.

Nous verrons dans la section 6.16 qu'un contrat noté co est composé de deux principales parties, une entête et un ensemble de clauses. Parmi les éléments de l'entête d'un contrat on a l'identifiant de l'agent noté ag. Ainsi, la notation $co.ag$ utilisée ci-dessous permet d'obtenir l'identifiant de l'agent auquel correspond un contrat.

$$\begin{aligned} &\forall ag \in \mathcal{A},\ \forall igr \in IG, \\ &\quad ismember(ag, igr) \iff \big(\exists co \in Co,\ co.ag = ag \wedge co \in coGrp(igr)\big) \\ &\qquad\qquad \vee \big(\exists igr' \in isubGrp(igr),\ \exists co' \in Co, \\ &\qquad\qquad\quad co'.ag = ag \wedge co' \in coGrp(igr')\big) \end{aligned} \tag{6.3}$$

Remarques

1. Afin de simplifier le calcul de l'appartenance d'un agent membre de igr', instance de groupe, aux instances de groupes englobants de igr', nous avons choisi de créer un contrat pour cet agent dans chacune de ces instances de groupe. Dans de tels contrats, seules les informations relatives à l'entête de contrat sont renseignées. En effet, l'agent étant implicitement membre du groupe parent, il n'y a donc pas lieu de mettre une clause. Ainsi selon la définition 6.16 d'un contrat présentée dans la section 6.1.6, la fonction $ismember(ag, igr')$ est défi-

nie comme suit :

$$\begin{aligned}
&\forall ag \in \mathcal{A},\ \forall igr, igr' \in IG,\ igr' \in isubGrp(igr) \\
&ismember(ag, igr') \iff \exists co \in Co,\ co.ag = ag \wedge co \in coGrp(igr') \\
&\qquad\qquad \wedge \big(\exists co' \in Co,\ co'.ag = ag \wedge co' \in coGrp(igr), \\
&\qquad\qquad\qquad \wedge\ co'.ContractantClauses = \phi \\
&\qquad\qquad\qquad \wedge\ co'.ContracteeClauses = \phi\big) \qquad (6.4)
\end{aligned}$$

2. Les ensembles $co'.ContractantClauses$ et $co'.ContracteeClauses$ sont vides à l'instant t où le contrat co' est créé. Ils pourront être différents de l'ensemble vide plus tard à un instant $t' > t$ si l'agent ag adopte des rôles, missions et buts dans une instance igr qui gère co'. Ces adoptions donneront alors lieu à la création de clauses qui seront enregistrés dans $co'.ContractantClauses$ et/ou $co'.ContracteeClauses$.

Exemple : Considérons la figure 6.2 qui représente trois instances de groupes $group_0, group_1, group_2$, ainsi que deux instances de portails $portal_1, portal_2$. Les instances $group_1, group_2$ sont des instances de sous-groupes de $group_0$. On suppose que (1) $portal_1$ gère les entrées et les sorties de $group_2$ pour les portes $gate_6, gate_7$. (2) $portal_2$ gère les entrées et les sorties de $group_0$ pour les portes $gate_1, gate_2$. (3) $portal_2$ gère également les entrées et les sorties de $group_1$ pour les portes $gate_3, gate_4$. Pour les processus d'entrée, à chaque porte correspond un appel à candidature qui n'est pas représenté sur la figure.

- L'$agent_1$ est membre de $group_0$ suite à réussite de la procédure d'entrée auprès de $portal_2$ pour la porte $gate_1$. Cependant il n'est ni membre de $group_1$ ni membre de $group_2$.
- L'$agent_2$ est membre de $group_1$ suite à la réussite de la procédure d'entrée auprès de $portal_2$ pour la porte $gate_4$. Puisque $group_1$ est un sous-groupe de $group_0$, il est aussi membre de $group_0$ mais il n'est pas membre de $group_2$.
- L'$agent_3$ est membre de $group_0$ suite à la réussite de la procédure d'entrée auprès de $portal_2$ pour la porte $gate_2$. Il est également membre de $group_2$ suite à la réussite de la procédure d'entrée auprès de $portal_1$ pour la porte $gate_26$. Il n'est pas membre de $group_1$.

Remarque : Lorsque deux instances de groupes gr_1 et gr_2 ne sont pas liées par une relation de sous-groupe, le passage d'une instance de groupe à une autre dans une

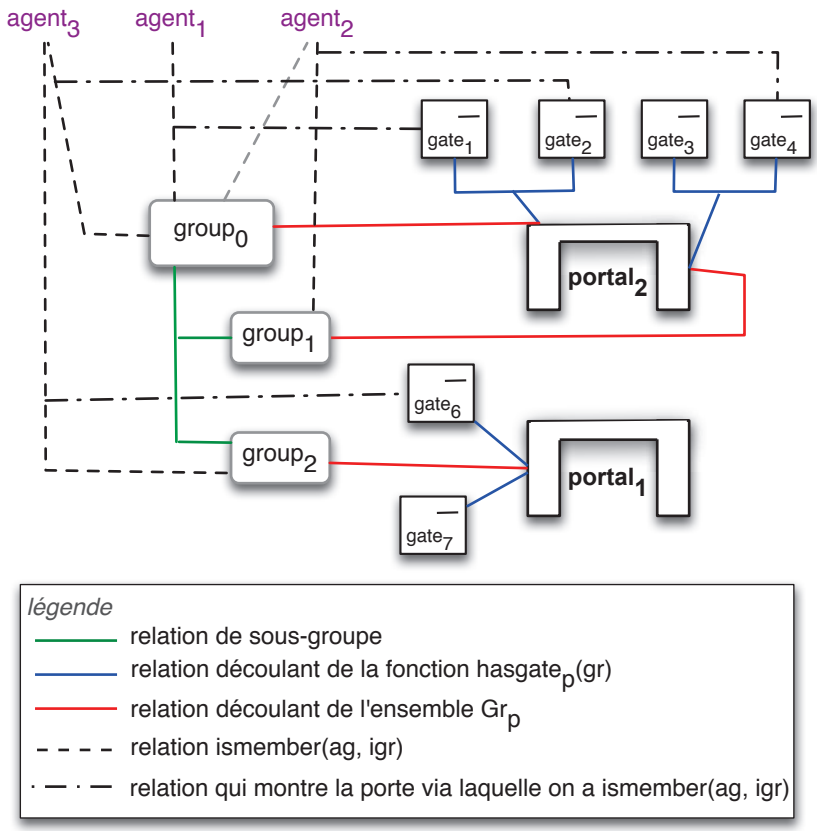


FIGURE 6.2 – illustration des cas d'entrées valides dans des groupes

entité organisationnelle est fonction des procédures de gestion des entrées de chaque instance groupe et donc des instances de portails qui leurs sont associées. Ainsi, en considérant l'exemple de la figure 6.2 l'$agent_3$ pour être membre de $group_1$ doit réussir une procédure d'entrée auprès de $portal_2$ pour l'une des portes de $group_1$. De même, si l'$agent_1$ qui est déjà dans $group_0$ veut entrer dans $group_1$, qui est associée à la même instance de portail $portal_2$ que $group_0$, alors il doit réussir une procédure d'entrée auprès de $portal_2$ pour l'une des portes de $group_1$.

6.1.3 Instance de schéma social

Une instance de schéma social d'une entité organisationnelle représente le cadre d'exécution de missions et de buts qui sont définies dans sa spécification. En effet, de la même façon qu'une instance de groupe est créée à partir d'une spécification de

groupe définie dans la spécification structurelle, une instance de schéma social est également créée à partir d'une spécification de schéma social définie dans la spécification fonctionnelle. Les missions gérées par une instance de schéma social sont composées de buts que doivent réaliser les agents. Pour participer à la réalisation des buts associés aux missions d'une instance de schéma social, les agents doivent d'abord s'engager aux missions concernées. Pour qu'un agent puisse s'engager à une mission d'une instance de schéma social, il doit être membre d'au moins une des instances des groupes responsables de l'instance de schéma social. Les missions auxquels peuvent s'engager un agent sont celles qui sont associées aux clauses de son (ses) contrat(s).

La gestion des missions par une instance de schéma social consiste d'une part à assurer le respect des contraintes de cardinalité définies sur les missions dans la spécification du schéma social (Cf.Équation 5.10). D'autre part, à maintenir l'état de chaque mission relativement à chaque clause de contrat concernée par cette mission. L'état des missions est accessible par la fonction $missionStatus$ présentée ci-dessus (Cf. Équation 6.2).

En outre, un agent qui s'est engagé à une mission peut l'abandonner par la suite. Ce qui entraîne la gestion des exigences d'abandons associées cette mission ainsi qu'aux buts concernées. Cf. Équation 5.17 et 5.18. Nous reviendrons sur la gestion des exigences d'abandons dans la section 6.2.2.

Quant à la gestion des buts, comme nous l'avons dit ci-dessus dans le paragraphe présentant la fonction $goalStatus$, elle consiste en la coordination des coopérations des agents en vu de la réalisation du but racine du schéma social. C'est cette gestion qui permet la mise à jour de l'état des normes relatives aux clauses des contrats. L'état des différentes normes est obtenu par la fonction $normStatus$ de l'équation 6.2).

6.1.4 Instance de portail

Une instance de portail d'une entité organisationnelle gère soit des entrées, soit des sorties soit des entrées et des sorties d'agents pour une ou plusieurs instance(s) de groupe(s). Les instances de groupes pour lesquelles une instance de portail gère des fonctions d'entrée / sortie est obtenue par la fonction $iporIG$ de l'équation 6.2. Le type de fonction (*entrée, sortie, entrée/sortie*) que gère une instance de portail pour une instance de groupe est déterminée par la spécification du portail et le type de l'instance de groupe.

En effet, de la même façon qu'une instance de groupe ou une instance de schéma social est créée à partir d'une spécification, une instance de portail est également créée à partir d'une spécification de portail définie dans l'EES. D'après l'équation 5.21, la

fonction $type_p(gr)$ permet de définir le type de fonction que devra gérer le portail p pour un type de groupe gr. Ainsi, pour une instance de groupe igr, on détermine son type avec la fonction $igrType$ de l'équation 6.2, l'instance ou les instances de portails qui gèrent ses fonctions d'entrée / sortie par la fonction $iporIG$ de l'équation 6.2. Pour chaque instance de portail $ipor$ de type $iporType(ipor) = p$ on détermine la fonction gérée par la la fonction $type_p(gr)$ de la spécification de l'instance de portail.

La gestion des entrées d'agents pour une instance de groupe, consiste en la gestion d'un sous-ensemble d'appels à candidatures. L'ensemble des appels à candidature gérées par une instance de portail s'obtient par la fonction $iporCa(ipor)$. L'état des différents appels à candidatures gérés par une instance de portail est obtenu par la fonction $missionStatus$ présentée ci dessus (Cf. équation 6.2).

Quant à la gestion des sorties, elle consiste en la gestion de processus de sortie que nous présentons dans la section 6.2.2.

Nous détaillons la notion d'appel à candidature dans la section suivante.

6.1.5 Appels à candidatures d'agents

Nous avons défini la notion d'appel à candidature d'agents (*CFAC : Call For Agent's Candidatures*) pour représenter de façon explicite, l'expression de chaque besoin précis que représente une porte définie dans un portail et qui s'adresse à des agents. En effet, un CFAC fournit toutes les informations qui permettent de gérer un processus d'entrée dans une instance de groupe. Un CFAC est créé pour une seule porte ga et est géré dans une seule instance de portail $ipor$.

Ainsi, les éléments d'un appel à candidature sont présentés par l'équation 6.5.

$$\forall ca \in Ca,$$
$$ca = \langle\, ga_{ca}, igr_{ca}, Entry\mathcal{R}eqs_{ca}, Adopt\mathcal{R}eqs_{ca}, nMin_{ca}, nMax_{ca}, timeout_{ca}\rangle \tag{6.5}$$

- ga_{ca} : identifiant de la porte à laquelle correspond l'appel à candidature, $ga \in Ga_p,\ p = porType(ipor)$. $ipor$ est l'instance de portail qui gère l'appel à candidature ca.
- igr_{ca} : est identifiant de l'instance de groupe associée à l'appel à candidature. Sa valeur est fournie par la fonction $caIgrp$ de l'équation 6.2.
- $Entry\mathcal{R}eqs_{ca}$: ensemble des exigences d'entrée associées à l'instance de portail $ipor$. $EntryReq_{ca} = EntryReq_p$, où $p = iporType(ipor)$.

- $AdoptRe\mathcal{q}s_{ca}$: ensemble de toutes les exigences d'adoption associées à ga_{ca}. Nous verrons dans le paragraphe suivant comment sont calculés les éléments de cet ensemble.
- $nMin_{ca}$: nombre minimum d'agent(s) qui pourront être admis pour cet appel à candidature. Si après l'analyse des candidatures, le nombre d'agent(s) retenu est inférieur à $nMin_{ca}$, l'appel à candidatures d'agents peut être relancé pour pouvoir admettre d'autres agents.
- $nMax_{ca}$: nombre maximum d'agent(s) admissible(s) pour cet appel à candidature.
- $timeout_{ca}$: date de fin de validité de l'appel à candidature.

Calcul des exigences d'adoption d'un CFAC et des valeurs $nMin_{ca}$ et $nMax_{ca}$

Nous avons vu dans le chapitre précédent que dans une OS, on peut définir des exigences d'adoption (AdoptReq) pour des rôles, des missions, et des buts.

D'après l'équation 5.22, une porte ga est toujours associée à une cible $target_{ga}$ qui est un triplet composé d'un rôle, d'une mission et d'un but. Ainsi, le calcul des exigences d'adoption d'un appel à candidature consiste en l'union des exigences d'adoption des éléments que comprend la cible de la porte ga_{icfac}.

$$AdoptRe\mathcal{q}s(ca) = adoptReq(target_{ga_{ca}}) \tag{6.6}$$

Nous présentons ci-dessous l'expression de la fonction $adoptReq$ selon que la cible $target_{ga_{ca}}$ est composée seulement d'un rôle, d'un rôle et d'une mission ou d'un rôle, d'une mission et d'un but.

L'expression de $adoptReq$ pour chaque forme de la cible sera suivie de celles de $nMin_{ca}$ et $nMax_{ca}$ car ces expressions dépendent également des éléments qui composent la porte de l'appel à candidature.

Nous rappelons que les valeurs min_r et max_r utilisées ci-dessous sont définies dans la spécification du rôle r associé à la cible de l'appel à candidature. Ces valeurs correspondent respectivement à la cardinalité minimale et maximale du rôle r. De même, les valeurs min_m, max_m, min_g et max_g également utilisées ci-dessous sont respectivement définies dans la spécification de la mission m et du but g associé à la cible de l'appel à candidature. Ces valeurs correspondent respectivement à la cardinalité minimale et maximale de la mission m et du but g.

1. $target_{ga_{ca}} = (r, \_, \_)$

$$adoptReq(target_{ga_{ca}}) = AdoptReq_r \tag{6.7}$$
$$nMin_{ca} = min_r \tag{6.8}$$
$$nMax_{ca} = max_r \tag{6.9}$$

2. $target_{ga_{ca}} = (r, m, \_)$

$$adoptReq(target_{ga_{ca}}) = AdoptReq_r \cup AdoptReq_m \tag{6.10}$$
$$nMin_{ca} = min(min_r, min_m) \tag{6.11}$$
$$nMax_{ca} = min(max_r, max_m) \tag{6.12}$$

3. $target_{ga_{ca}} = (r, m, g)$

$$adoptReq(target_{ga_{ca}}) = AdoptReq_r \cup AdoptReq_m \cup AdoptReq_g \tag{6.13}$$
$$nMin_{ca} = min(min_r, min_m, min_g) \tag{6.14}$$
$$nMax_{ca} = min(max_r, max_m, min_g) \tag{6.15}$$

Remarque : Considérons à nouveau l'exemple de la figure 6.2.

- Si l'$agent_3$ fait une démarche d'entrée dans $group_1$ auprès de $portal_2$, il doit satisfaire les exigences d'entrée et d'adoption relatives à la procédure d'entrée.
- Si $agent_1$ qui est déjà dans $group_0$ fait une démarche d'entrée dans $group_1$ auprès de $portal_2$, le processus d'entrée ne vérifiera plus les exigences d'entée de $portal_2$ puisque celles-ci ont été vérifiées lors de son entrée dans $group_0$. Seules les exigences d'adoption seront vérifiées.

Exemples : la spécification d'entrée / sortie de l'OS de l'exemple writePaper présenté dans le chapitre précédant était constituée d'un seul portail $wpPor$ définie comme suit :

- $wpPor = \langle Grp_{wpPor}, Ga_{wpPor}, EntryReqs_{wpPor}, ExitReqs_{wpPor}, type_{wpPor}, hasgate_{wpPor}, \rangle$
  - $Grp_{wpPor} = \{wpgroup\}$
  - $Ga_{wpPor} = \{ga_1, ga_2\}$
  - $hasgate_{wpPor}(wpgroup) = \{ga_1, ga_2\}$
  - $EntryReqs_{wpPor} = \{en\_req_1\}$
    $en\_req_1 = \langle EntryReq, hasSkill == MAS, mandatory \rangle$

  - $ExitReqs_{wpPor} = \phi$
  - $type_{wpPor}(wpgroup) = EntryExit$
- $ga_1 = \langle target_{ga_1}, 5\ day \rangle, \quad target_{ga_1} = (writer, \_, \_)$
- $ga_2 = \langle target_{ga_2}, 6\ day \rangle, \quad target_{ga_2} = (editor, mMan, \_)$

Les identifiants ga_1, ga_2 sont ceux des portes d'entrée dans le groupe $wpgroup$. La spécification du rôle $writer$ présentée dans la section 5.3.4 est celle de la mission $mMan$ présentée dans la section 5.4.4 sont reprises ci-dessous afin de faciliter la compréhension des appels à candidatures $cfac_1$ et $cfac_2$ correspondants à ga_1 et ga_2 présentés ci-dessous.

- $writer = \langle AdoptReq_{writer}, \phi \rangle$
  - $AdoptReq_{writer} = \{ad\_req_{01}, ad\_req_{02}\}$
  - $ad\_req_{01} = \langle AdoptReq, DomainSkill == MAS\_organisation, mandatory \rangle$
  - $ad\_req_{02} = \langle AdoptReq, LanguageSkill == English, mandatory \rangle$
- $editor = \langle AdoptReq_{editor}, \phi \rangle$
  - $AdoptReq_{editor} = AdoptReq_{writer} = \{ad\_req_{01}, ad\_req_{02}\}$
- $mMan = \langle AdoptReq_{mMan}, LeaveReq_{mMan} \rangle$
  - $AdoptReq_{mMan} = \{ad\_req_{03}\}$
  - $LeaveReq_{mMan} = \{le\_req_{11}\}$
  - $ad\_req_{03} = \langle AdoptReq, hasCapability == Management, optional \rangle$
  - $le\_req_{11} = \langle AdoptReq, hasFinishedAllGoal == \_, mandatory \rangle$

On note $wpigrp_1$ une instance de groupe de $wpgroup$ et $wpipor_1$ une instance de portail de $wpPor$. On suppose que $cfac_1$ et $cfac_2$ sont gérés par $wpipor_1$ pour $wpigrp_1$.

1. $cfac_1 = \langle ga_1, iwpgrp_1, AdoptReqs_{cfac_1}, EntryReqs_{cfac_1}, 1, 5, 4day \rangle$
   $EntryReqs_{cfac_1} = \{en\_req_1\}$
   $AdoptReqs_{cfac_1} = AdoptReq_{writer} = \{ad\_req_{01},\ ad\_req_{02}\}$
2. $cfac_2 = \langle ga_2, iwpgrp_1, EntryReq_{cfac_2}, AdoptReqs_{cfac_2}, 1, 5, 6day \rangle$
   $EntryReq_{cfac_2} = \{en\_req_1\}$
   $AdoptReqs_{cfac_2} = AdoptReq_{editor} \cup AdoptReq_{mMan}$
   $= \{ad\_req_{01},\ ad\_req_{02},\ ad\_req_{03}\}$

6.1.6 Contrat

Dans une entité organisationnelle oe, un contrat est créé par une instance de portail à l'issu de la réussite du processus d'entrée d'un agent ag.

Nous avons défini un modèle de contrat composé de deux principaux composants : une entête et un ensemble de clauses. Ainsi

$$\forall co \in Co,\ co = \langle Header, ContractantClauses, ContracteeClauses \rangle \quad (6.16)$$

Entête de contrat

L'entête d'un contrat, notée *Header* fournit les informations générales relatives à l'enregistrement de l'entrée d'un agent dans une instance de groupe : identifiant des parties signataires du contrat, date d'entrée, etc.

$$Header = \langle ag, igr, orgAgent, entranceDate, exitDate \rangle$$

- ag : identifiant de l'agent à qui correspond le contrat.
- igr : identifiant de l'instance de groupe dans laquelle sera géré le contrat.
- $orgAgent$: identifiant de l'agent organisationnel ayant validé la création du contrat.
- $entranceDate$: date à laquelle le contrat est créé.
- $exitDate$: date de sortie de l'agent de igr.

Après avoir décrit les éléments de l'entête d'un contrat, nous allons voir la description d'une clause.

Clauses d'un contrat

Les ensembles *ContractantClauses* et *ContracteeClauses* d'un contrat contiennent respectivement la liste de toutes les clauses qui représentent les engagements d'un agent vis-à-vis de l'instance de groupe de l'oe dans laquelle sera enregistré son contrat et la liste de toutes les engagements de l'entité organisationnelle oe vis-à-vis de l'agent. Chaque clause correspond à une norme définie dans la spécification normative (NS) de l'OS.

La représentation et la description des éléments d'une clause sont présentées ci-dessous.

$$\begin{aligned} &\forall cl \in ContractantClauses \cup ContracteeClauses, \\ &cl = \langle r_{cl}, n_{cl}, m_{cl}, isch_{cl}, Goals_{cl}, date_{cl}, orgAgent_{cl} \rangle \end{aligned} \quad (6.17)$$

- r_{cl} : identifiant du rôle de la cible de l'appel à candidature ca suite auquel la clause de contrat est créée : $r_{cl} = r_{ca}$.
- n_{cl} : identifiant de la norme qui régit la clause. Cet identifiant est une valeur de l'ensemble N (Cf. équation 5.19) des normes définies dans la spécification normative et l'identifiant du rôle associé à n_{cl} doit être égale à r_{cl} : $n_{cl} \in N, roleN(n_{cl}) = r_{cl}$.
- m_{cl} : identifiant de la mission sur laquelle porte la clause. La valeur de m_{cl} est celle associée à la norme n_{cl}, $m_{cl} = missionN(n_{cl})$.
- $isch_{cl}$: identifiant de l'instance de schéma social dans laquelle est définie et gérée la mission de la clause m_{cl}.
- $Goals_{cl}$: ensemble de tous les buts sur lesquels porte la clause,
  $\forall g \in Goals_{cl}, g \in mo_s(m_{cl})$. La fonction mo_s : est définie dans l'équation 5.10 elle fournit l'ensemble des buts d'une mission définie dans un schéma social s. La valeur de s ici est obtenue par la fonction $schType(isch_{cl})$ (Cf. équation 6.2).
- $date_{cl}$: date à laquelle la clause est établie.
- $orgAgent_{cl}$: identifiant de l'agent organisationnel qui a validé la clause.

Remarques

1. De façon générale, les clauses d'un contrat sont créées selon la forme de la cible $target_{ga}$ de la porte ga_{ca} correspondante à l'appel à candidature ca. Ainsi, selon les trois formes de la cible d'une porte définie dans la section 5.6.2, on a les deux cas suivants :
   - $target_{ga} = (r, \_, \_)$: l'ensemble $ContractantClauses$ est constitué d'autant de clause(s) que de norme(s) n_i dans lesquelles le rôle r est associé à la mission de n_i par un $type$ (Cf. représentation d'une norme : Équation 5.19). La valeur de $type$ appartient à l'ensemble $\{obligation, permission, forbidden\}$.
     L'ensemble $ContracteeClauses$ est constitué d'autant de clause(s) que de norme(s) n_j auxquels le rôle r est associé à la condition de n_j.
     L'exemple de contrat présenté ci-dessous illustre cette remarque.

   - $target_{ga} = (r, m, \_)$ ou $target_{ga} = (r, m, g)$: Dans l'ensemble $ContractantClauses$, on a une clause correspondante à la norme n_i dans laquelle le rôle r qui est associé à la mission m par un $type$ dont la valeur appartient à l'ensemble $\{obligation, permission, forbidden\}$.
     Dans l'ensemble $ContracteeClauses$ on a autant de clause(s) que de norme(s) n_j auxquels le rôle r est associé à la condition de n_j.

2. Les arguments $date$ et $orgAgent$ d'une clause sont utiles particulièrement lorsque de nouvelles clauses sont ajoutées à un contrat déjà existant. En effet, l'entête d'un contrat comprend déjà une date et l'identifiant d'un $OrgAgent$. Ceux-ci correspondent respectivement à la date de la création d'un contrat, et à l'identifiant de l'agent organisationnel ayant validé la création du contrat.
   Par ailleurs, nous considérons qu'au cours du cycle de vie d'un agent dans une entité organisationnelle, celui-ci peut adopter d'autres rôles et donc s'engager à d'autres missions et buts. Ces nouveaux engagements pourront être enregistrés dans son contrat. Raison pour laquelle, nous avons associé à chaque clause une date qui identifie la date de création ou d'enregistrement de la clause et l'identifiant de l'OrgAgent ayant validé la clause.

Exemple de contrat

On considère dans l'exemple de contrat $contract_1$ ci-dessous qu'un agent ayant pour identifiant $agent_1$ a été admis dans l'instance de groupe $wpigrp_1$ pour l'appel à candidature $cfac_1$ qui concerne le rôle $writer$. On suppose que $wpigrp_1$ est responsable des instances de schémas sociaux ayant pour identifiant respectif $wpisch_1$ et $wprewisch_1$ dans laquelle sont gérés les missions des différentes clauses de ce contrat. Les 4 clauses du contrat correspondantes aux normes n_2, n_3, n_4 et n_5 (Cf. Tableau 5.1) de la cible $target_{cfac1} = (writer, \_, \_)$.

$$
\begin{aligned}
co_1 &= \langle Header, ContractantClauses, ContracteeClauses\rangle \\
&\quad Header = \langle agent_1, wpigrp_1, OrgAgent_5, 05/05/2011, dd/mm/yyyy\rangle \\
&\quad ClausesContractant = \{cl_1, cl_2\} \\
&\qquad cl_1 = \langle writer, n_2, mCol, wpisch_1, Goals_{cl_1}, 05/05/2011, OrgAgent_5\rangle \\
&\qquad\quad Goals_{cl_1} = \{wsecs\} \\
&\qquad cl_2 = \langle writer, n_3, mBib, wpisch_1, Goals_{cl_2}, 05/05/2011, OrgAgent_5\rangle \\
&\qquad\quad Goals_{cl_2} = \{wref\} \\
&\quad ClausesContractee = \{cl_3, cl_4\} \\
&\qquad cl_3 = \langle writer, n_4, mRew1, wprewisch_1, Goals_{cl_3}, 05/05/2011, OrgAgent_5\rangle \\
&\qquad\quad Goals_{cl_3} = \{rewCol\} \\
&\qquad cl_4 = \langle writer, n_5, mRew2, wprewisch_1, Goals_{cl_4}, 05/05/2011, OrgAgent_5\rangle \\
&\qquad\quad Goals_{cl_4} = \{rewBib\}
\end{aligned}
$$

Les clauses cl_1 et cl_2 définissent les engagements de l'$agent_1$ vis-à-vis de l'organisation relativement à son rôle $writer$. Ces engagements concernent les normes n_2 et

n_3. Les missions $mCol$ et $mBib$ sont gérées par l'instance de schéma social $wpisch_1$. L'$agent_1$ aura donc l'obligation de réaliser le but $wsecs$ de la mission $mCol$ ainsi que le but $wref$ de la mission $mBib$.

Les clauses cl_3 et cl_4 définissent les engagements de l'organisation vis-à-vis de l'$agent_1$ relativement à son rôle $writer$. Ces engagements concernent les normes n_4 et n_5. Les missions $mRew1$, $mRew2$ respectives de ces normes sont gérées par l'instance de schéma social $wprewisch_1$. Selon la norme n_3 un agent jouant le rôle $editor$ aura l'obligation de récompenser l'$agent_1$ en réalisant le but $rewCol$ de la mission $mRew$ si la condition $fulfilled(n_2)$ de cette norme (n_4) est vérifiée. De même, si la condition $fulfilled(n_3)$ de la norme n_5 est vérifiée un agent jouant le rôle $editor$ aura l'obligation de récompenser l'$agent_1$ en réalisant le but $rewBib$ de la mission $mRew2$.

6.2 Gestion des entrées / sorties

Nous avons présenté dans la section ci-dessus les principaux composants d'une entité organisationnelle. Dans cette section, nous allons voir comment se déroulent les processus d'entrée / sortie qui permettent aux agents respectivement d'entrer dans une organisation et d'en sortir.

6.2.1 Gestion des entrées et adoptions

Un processus d'entrée d'agents dans une entité organisationnelle regroupe toutes les étapes que doivent suivre des agents lors d'une démarche d'entrée dans cette OE. Notre proposition de processus d'entrée comprend deux principales phases : une phase de candidature et une phase de contractualisation.

- La *phase de candidature* gère la création des appels à candidatures, les dépôts de candidatures d'agents, l'examen de(s) candidature(s) et la publication des agents pré-admis.
- Pendant la *phase de contractualisation*, les contrats des agents admis sont créés et enregistrés dans l'instance de groupe concernée par le processus d'entrée.

Pour la présentation de ces différentes phases dans les sections ci-dessous, nous considérerons que nous avons une entité organisationnelle décrite comme suit.

$$wpoe_1 = \langle OS,\ IG,\ IS,\ IP,\ Ca,\ Co,\ grpType,\ schType,\ porType,\ isubGrp,\ igrpIS,\ ischIG,\ iporIG,\ iporCa,\ caIgrp,\ igrpCo,\ cfacStatus,\ missionStatus,\ goalStatus,\ normStatus \rangle^t \quad (6.18)$$

- $OS = writePaper$: la spécification organisationnelle de $wpoe_1$ est celle de l'exemple write-paper dont nous avons présenté les éléments SS, FS, NS, EES dans le chapitre précédent.
- $IG = \{wpigrp_1\}$: l'entité organisationnelle $wpoe_1$ a une seule instance de groupe $wpigrp\_1$ et conformément à sa spécification, $wpigrp\_1$ n'a pas d'instance de sous-groupe. Elle est responsable d'une seule instance de schéma social notée $wpisch_1$
  - $grpType(iwpgrp) = wpgroup$.
  - $isubGrp(wpigrp_1) = \phi$.
  - $igrpIS(wpigrp_1) = \{wpisch_1\}$
- $IS = \{wpisch_1\}$: $schType(wpisch_1) = wpSch$. Une seule instance de groupe est responsable de $wpisch_1$ ($ischIG(wpisch_1) = \{wpigrp_1\}$).
- $Ca = \{cfac_1, cfac_2\}$: l'ensemble des appels à candidatures de $wpoe_1$. L'instance de groupe associé à chacun de ces appels à candidature est $wpigrp_1$.
  - $caIgrp(cfac_1) = wpigrp_1$
  - $caIgrp(cfac_2) = wpigrp_1$
- $IP = \{wpipor_1\}$: $porType(wpipor_1) = wpPor$. L'instance de portail $wpipor_1$ est associée à l'instance de groupe $wpigrp_1$ et gère pour cette instance de groupe les deux appels à candidature de l'ensemble Ca.
  - $iporIG(wpipor_1) = \{wpigrp_1\}$
  - $iporCa(wpipor_1) = \{cfac_1, cfac_2\}$

6.2.1.1 Candidatures

Dans une entité organisationnelle, les processus d'entrées sont gérés par les instances de portail. Un processus d'entrée concerne une porte qui est associée à une instance de groupe. La première phase d'un processus d'entrée comporte quatre étapes que nous détaillons dans les paragraphes ci-dessous.

Création des appels à candidatures

L'initialisation d'un processus d'entrée commence par la création d'un ensemble d'appels à candidatures. Nous avons présenté dans la section 6.1.5 les différents éléments d'un appel à candidature.

Une instance de portail qui est associée à une instance de groupe pour laquelle elle gère les entrées d'agents créée un appel à candidature pour chaque porte associée à cette instance de groupe.

Exemple : comme exemple on considèrera les appels à candidatures $cfac_1$ et $cfac_2$ de l'exemple *write-paper* présentés dans la section 6.1.5.

Remarque : Dans une entité organisationnelle oe, les appels à candidatures ont deux fonctions :

1. D'une part, ils permettent de gérer les entrées d'agents dans une instance de groupe. En d'autres termes, tout agent ag qui n'est pas encore membre d'une instance de groupe $igr \in IG$ –ag peut être membre d'une autre instance de groupe $igr' \neq igr$– et qui voudrait l'être devra répondre à un appel à candidature associé à l'instance de groupe pour pouvoir être admis.
2. D'autre part, il servent à gérer les adoptions de rôles d'une instance de groupe pour des agents déjà membres de celle-ci. En d'autres termes, un agent ag qui est déjà membre d'une instance de groupe igr –c'est-à-dire que ag joue au moins un rôle dans igr– peut répondre à des appels à candidatures concernant d'autres rôles de cette instance de groupe.

La distinction entre un agent qui répond à un appel à candidature ca et qui est déjà membre de l'instance de groupe igr associée à ca se fait à travers les données d'une candidature présentée dans le paragraphe ci-dessous.

Dépôts de candidatures d'agents

Les agents expriment leur volonté d'entrer dans une entité organisationnelle et plus précisément dans une instance de groupe igr en soumettant une candidature pour un appel à candidature ca publié auprès d'une instance de portail $ipor$ ($\in iporCa(ipor)$). Toute candidature d'agent correspond donc à un appel à candidature existant dans une instance de portail. L'enregistrement d'une candidature se fait auprès de l'instance de portail qui gère l'appel à candidature. La représentation d'une candidature est donnée par l'équation 6.19.

$$cand = \langle agent_{cand}, ca_{cand}, ERValues_{cand}, ARValues_{cand} \rangle \tag{6.19}$$

- $agent_{cand}$: identifiant de l'agent candidat.
- ca_{cand} : identifiant de l'appel à candidature correspondant à la candidature.
- $ERValues_{cand}$: liste de paires $(erId, erValue)$ où $erId$ est l'identifiant d'une exigence d'entrée et $erValue$ est la valeur fournie par l'agent candidat pour $erId$.
- $ARValues_{cand}$: liste de paires $(arId, arValue)$ où $arId$ est l'identifiant d'une exigence d'adoption et $arValue$ est la valeur fournie par l'agent pour $arId$.

Exemples :

1. $cand_1 = \langle agent_1, cfac_1, ERValues_{cand_1}, ARValues_{cand_1} \rangle$
   - $ERValues_{cand_1} = \{(en\_req_1 : hasSkill\,;\, MAS)\}$
   - $ARValues_{cand_1} = \{(ad\_req_{01} : DomainSkill\,;\, MAS\_organisation),$ $(ad\_req_{02}, LanguageSkill\,;\, English)\}$
2. $cand_2 = \langle agent_2, cfac_1, ERValues_{cand_2}, ARValues_{cand_2} \rangle$
   - $ERValues_{cand_2} = \{(en\_req_1 : hasSkill\,;\, MAS)\}$
   - $ARValues_{cand_2} = \{(ad\_req_{01} : DomainSkill\,;\, MAS\_organisation),$ $(ad\_req_{02} : LanguageSkill\,;\, French)\}$
3. $cand_3 = \langle agent_3, cfac_1, ERValues_{cand_3}, ARValues_{cand_3} \rangle$
   - $ERValues_{cand_3} = \{(en\_req_1 : hasSkill\,;\, MAS)\}$
   - $ARValues_{cand_3} = \{(ad\_req_{01} : DomainSkill\,;\, MAS\_environment,$ $MAS\_organisation), (ad\_req_{02} : LanguageSkill\,;\, English)\}$
4. $cand_5 = \langle agent_5, cfac_2, ERValues_{cand_5}, ARValues_{cand_5} \rangle$
   - $ERValues_{cand_5} = \{(en\_req_1 : hasSkill\,;\, MAS)\}$
   - $ARValues_{cand_5} = \{(ad\_req_{01} : DomainSkill\,;\, MAS\_organisation),$ $(ad\_req_{02} : LanguageSkill\,;\, French),$ $(ad\_req_{03} : hasCapability\,;\, Management)\}$
5. $cand_6 = \langle agent_6, cfac_2, ERValues_{cand_6}, ARValues_{cand_6} \rangle$
   - $ERValues_{cand_6} = \{(en\_req_1 : hasSkill\,;\, MAS)\}$
   - $ARValues_{cand_6} = \{(ad\_req_{01} : DomainSkill\,;\, MAS\_organisation),$ $(ad\_req_{02} : LanguageSkill\,;\, English),$ $(ad\_req_{03} : hasCapability\,;\, Management)\}$

Remarques

- Les candidatures peuvent être soumises et enregistrées pendant toute la durée $timeout_{ca}$ de publication d'un appel à candidatures. Dès que le $timeout_{ca}$ est totalement écoulé, l'étape d'examen de candidatures peut commencer.

– Cependant, pour certaines *portes*, la valeur de $timeout_{ca}$ peut être égale à l'infini ($timeout_{ca} = \infty$). Par exemple, dans une organisation ouverte de type e-commerce, le nombre de vendeur et d'acheteur n'est généralement pas limité. Ainsi, le $timeout$ d'appel à candidatures pour ces rôles est généralement égal à l'infinie. Avec ce type de d'appel à candidature, l'examen de chaque candidatures se fait dès qu'elle est enregistrée.

Examen des candidatures

L'examen des candidatures d'un processus d'entrée consiste d'une part à vérifier que les exigences requises pour être admis dans l'instance de groupe et pour adopter le rôle, la (ou les) mission(s) et ainsi que le (ou les) but(s) de la cible de l'appel à candidatures sont satisfaites.

D'autre part puisque des agents déjà membres de l'instance de groupe associée à l'appel à candidature peuvent également candidater, et que des contraintes de compatibilités existent entre des rôles d'une instance de groupe alors, la vérification de ces contraintes est réalisée dans cette étape du processus d'entrée.

La représentation d'un appel à candidature ca (Cf. Équation 6.5) comprend également deux variables $nMin_{ca}, nMax_{ca}$ représentant respectivement le nombre minimum et maximum d'agent(s) pouvant être admis pour ce ca. Ainsi, cette étape comporte également la vérification de chacune de ces valeurs en particulier celle de $nMax_{ca}$. L'analyse des candidatures d'agents pour un CFAC d'un processus d'entrée comprend donc les phases suivantes :

1. **Vérification de la contrainte de cardinalité maximale d'un** ca : qui consiste à s'assurer que le nombre d'agents déjà admis pour l'appel à candidature ca n'est pas encore égal à la valeur maximale c'est-à-dire à $maxAgent$.
2. **Vérification des contraintes de compatibilités** : dans la section 5.3.3, nous avons présenté la notion de contrainte de compatibilité qui est défini entre des rôles d'un groupe. Celle-ci permet de garantir une cohérence structurelle entre les agents jouant des rôles dans les instances de groupes. Puisque les contraintes de compatibilités ont deux types de portées : $intra - group$ et $inter - group$, nous avons deux cas de vérification de ces contraintes.
   – ***Vérification des contraintes de compatibilité intra-groupe***
   Elle concerne toute candidature pour laquelle l'agent est déjà membre de l'instance de groupe associée à l'appel à candidatures. Rappelons que lorsqu'un agent est membre d'une instance de sous-groupe igr' d'une instance de groupe igr alors il est également membre de igr.

La vérification de compatibilité pour une candidature ca consiste dans ce cas à vérifier dans igr si le rôle de la cible associée à la porte de l'appel à candidature est compatible avec le ou les autre(s) rôle(s) déjà joué(s) par l'agent associé à la candidature ca.
L'algorithme 1 montre les instructions de traitement de cette vérification auprès d'une instance de groupe.

Algorithm 1 verifyCompatibilityConstraint(igr, agentCand, roleCand)

```
if (ismember(agentCand, igr)) then
  co = Co_igr.agentCand
  while (co.ContractantClauses.hasNext()) do
    cl = nextContractantClauses(co)
    if (NOT compatible(cl.role, roleCand)) then
      return false
    end if
  end while
  if (hasSuperGrp(igr)) then
    igr' = superGrp(igr)
    if (NOT verifyCompatibilityConstraint(igr', agentCand, roleCand)) then
      return false
    end if
  end if
  while ( isubGrp(igr).hasNext()) do
    igr" = isubGrp(igr).next()
    if (NOT verifyCompatibilityConstraint(igr", agentCand, roleCand)) then
      return false
    end if
  end while
end if
return true
```

- Il prend en paramètre l'identifiant de l'instance de groupe igr dans laquelle se fait la vérification, l'identifiant de l'agent candidat $agentCand$ et l'identifiant du rôle $roleCand$ concerné par la candidature.

- La première instruction vérifie si l'agent est déjà membre de igr. Elle utilise la fonction $ismember$ définie par l'équation 6.3 pour cette vérification.

- Dans le cas où l'agent est membre de igr, l'algorithme vérifie que pour chaque clause appartenant à l'ensemble des clauses contractante, le rôle associé est compatible avec le rôle concerné par la candidature ($roleCand$).

- Si cette condition n'est pas vérifiée pour l'une des clauses l'algorithme s'arrête et retourne la valeur FAUX (**false**).

- Si cette condition est vérifiée pour toutes les clauses contractantes de l'agent dans igr et si igr est sous-groupe d'une instance de groupe igr', alors la vérification se fait également dans l'instance de groupe père igr'.

- Si à la suite des deux étapes de vérification précédente la condition est toujours vérifiée, alors la vérification se poursuit dans les instances de sous-groupes que nous notons igr'' de igr par appel récursif de l'algorithme.

– ***Vérification des contraintes de compatibilité inter-groupe.***
Elle concerne toute candidature pour laquelle l'agent est déjà membre de l'instance de groupe igr associée à l'appel à candidatures ou de toute autre instance igr' de même type que igr. Dans ce cas, l'algorithme de vérification de compatibilité 1 est exécuté dans chaque instance de groupe du type de igr.

3. **Vérification des exigences** : à ce niveau de l'examen d'une candidature, l'objectif est de faire un appariement entre les exigences spécifiées pour un appel à candidature ca et les valeurs fournies par un agent dans sa candidature $cand$. Cette vérification permet de déterminer si les valeurs fournies dans une candidature pour chaque exigence correspondent aux valeurs escomptées par l'entité organisationnelle.

Exemple : pour la vérification des éléments des ensembles $ERValues_{cand}$ et $ARValues_{cand}$ nous avons défini une fonction $check(req_{id}, cand)$ qui retourne une valeur booléenne *true* (T) si la (ou les) valeur(s) fournie(s) par le candidat $cand$ pour l'exigence req_{id} correspond(ent) à (ou aux) valeur(s) escomptée(s). Dans le cas contraire $check(req_{id}, cand)$ retourne *false* (F).

Pour les exemples de candidatures ci-dessus présentées, nous avons les résultats suivants.

Candidatures pour $cfac_1$

$cand_1$

$ERValues_{cand_1}$

$check(en\_req_1, cand_1) = T$

$ARValues_{cand_1}$

$check(ad\_req_{01}, cand_1) = T$

$check(ad\_req_{02}, cand_1) = T$

$cand_2$

$ERValues_{cand_2}$

$check(en\_req_1, cand_2) = T$

$ARValues_{cand_2}$

$check(ad\_req_{01}, cand_2) = T$

$check(ad\_req_{02}, cand_2) = F$

$cand_3$

$ERValues_{cand_3}$

$check(en\_req_1, cand_3) = T$

$ARValues_{cand_3}$

$check(ad\_req_{01}, cand_3) = T$

$check(ad\_req_{02}, cand_3) = T$

Candidatures pour $cfac_2$

$cand_4$

$ERValues_{cand_4}$

$check(en\_req_1, cand_4) = T$

$ARValues_{cand_4}$

$check(ad\_req_{01}, cand_4) = T$

$check(ad\_req_{02}, cand_4) = F$

$check(ad\_req_{03}, cand_4) = T$

$cand_5$

$ERValues_{cand_5}$

$check(en\_req_1, cand_5) = T$

$ARValues_{cand_5}$

$check(ad\_req_{01}, cand_5) = T$

$check(ad\_req_{02}, cand_5) = T$

$check(ad\_req_{03}, cand_5) = T$

4. **Sélection des agents admis selon un critère d'admissibilité**. La propriété associée à chaque exigence détermine si cette exigence est obligatoire ($property = mandatory$) ou si elle est optionnelle ($property = optional$). Ainsi toute candidature est admissible si elle fournit pour toutes les exigences de propriété $mandatory$ une valeur satisfaisante. Pour les candidatures qui satisfont cette première contrainte d'admissibilité, un *critère d'admissibilité* est défini afin de permettre de classer et sélectionner les agents définitivement admis.

Exemple : considérons le critère d'admissibilité suivant : *Ordre décroissant du nombre d'exigences de type* $optional$ *fournies en plus de toutes les exigences de type* $mandatory$. Les candidatures ci-dessus présentées et examinées sont classées selon ce critère d'admissibilité comme suit :

Candidatures pour $cfac_1$	**Candidatures pour** $cfac_2$
1. $cand_3$	1. $cand_5$
2. $cand_1$	2. $cand_4$
3. $cand_2$	

NB : Les critères d'admissibilités pour des appels à candidature dans une entité organisationnelle peuvent être définis dans la spécification des missions et buts des agents organisationnels et ainsi faire partis de leurs stratégies de décisions.

La dernière étape du processus d'entrée est présentée dans la section qui suit.

6.2.1.2 Contractualisation

Après l'admission d'un agent dans une organisation, celui-ci peut confirmer ou infirmer sa volonté effective d'entrer dans l'organisation. Pour cela, il devra participer à la négociation et spécification des clauses de son contrat ainsi qu'à sa validation dans le cas où il souhaite confirmer son entrée. Dans le cas contraire, il pourra notifier son désistement à l'OrgAgent par l'envoi d'un message.

Négociation des clauses d'un contrat

La négociation des clauses d'un contrat dépend du type de la cyble $target$ associée à la *porte* d'un appel à candidature $cfac$ ayant donné lieu à la création des clauses du contrat. Ainsi, selon que $target$ est composé seulement d'un *rôle* ; ou d'un *rôle* et d'une *mission* ; ou encore d'un *rôle*, d'une *mission* et d'un *but*, la nature des engagements de contrat ne sont pas les même et donc la négociation ne porte pas sur les mêmes choses. Ainsi, on a les cas de négociation suivants selon la forme de $target$.

1. (r, m, g) : pas de négociation, car l'objectif de la clause est principalement le but g.
2. $(r, m, \_)$: la négociation peut se faire uniquement sur les buts de la mission m.

3. $(r, \_, \_)$: les négociations concernent les missions associées au rôle r via les normes de la NS.

Lorsque toutes les clauses d'un appel à candidatures sont négociées, celles-ci sont enregistrées dans le contrat de l'agent qui est soit créé dans la cas d'un processus d'entrée dans une instance de groupe, soit mis à jour dans le cas d'une adoption.

En effet, tout agent membre d'une instance de groupe a un seul et unique contrat qui a été créé lors de son entrée. Il peut adopter d'autres rôles dans cette instance de groupe et donc d'autres missions et buts de schémas sociaux dont cette instance de groupe est responsable. En cas d'adoption de nouveaux rôles, missions et buts relatives à une instance de groupe dont on est déjà membre, les nouvelles clauses relatives à ces adoptions sont ajoutées dans le contrat de l'agent concerné. Nous présentons les cas de modification d'un contrat dans la section qui suit.

Modification d'un contrat

Pour pouvoir avoir de nouvelles clauses dans son contrat au sein d'une instance de groupe, tout agent devra comme dans le cas d'une démarche d'entrée répondre à un appel à candidature associé à cette instance de groupe en soumettant une candidature avec les données requises. Ces candidatures seront traitées selon les étapes présentées dans la section 6.2.1.1.

Par ailleurs, un agent membre d'une instance de groupe d'une entité organisationnelle oe peut répondre à des appels à candidatures qui concernent d'autres instances de groupe de cette oe. En cas d'admission, un nouveau contrat sera créé dans les instances de groupes concernées si l'agent n'y a pas déjà un contrat.

En conséquence, un agent pourra avoir plusieurs contrats dans des instances de groupes différentes d'une même entité organisationnelle. Cette approche est justifiée par notre choix de gestion distribuée d'une entité organisationnelle. Elle offre l'avantage de décongestionner les traitements à réaliser par une seule entité dans le cas d'une centralisation.

Après l'admission d'un agent dans une instance de groupe et donc la création de son contrat, celui-ci peut vouloir abandonner certaines clauses ou sortir de l'instance de groupe. Nous présentons dans la section qui suit les processus de gestion de ces fonctionnalités.

6.2.2 Gestion des sorties et abandons

La gestion des abandons de clause de contrat se gère au sein d'une instance de groupe tandis que celle de sortie est gérée par une instance de portail. Dans les deux cas, le processus se décompose en trois phases : demande d'abandon ou de sortie, analyse de la demande, négociation et validation de l'abandon ou de la sortie.

6.2.2.1 Demande d'abandon ou de sortie

Tout agent ayant un contrat dans une instance de groupe peut abandonner des clauses de son contrat ou demander à sortir de l'instance de groupe. Une demande d'abandon ($leaveRequest$) est soumise à une instance de groupe tandis qu'une demande de sortie ($exitRequest$) est soumise à une instance de portail. $leaveRequest$ et $exitRequest$ sont respectivement définis comme suit :

$$leaveRequest = \langle agent_{lr}, cl_{lr}, date_{lr} \rangle \tag{6.20}$$

$$exitRequest = \langle agent_{er}, igr_{er}, date_{er} \rangle \tag{6.21}$$

- $agent_{lr}$ / $agent_{er}$: identifiant de l'agent qui fait la demande d'abandon ou de sortie.
- cl_{lr} : identifiant de la clause que l'agent veut d'abandonner.
- igr_{er} : identifiant de l'instance de groupe de laquelle l'agent veut sortir.
- $date_{lr}$ / $date_{er}$: date à laquelle la demande d'abandon ou de sortie est enregistrée.

Examen d'une demande d'abandon ou de sortie

L'analyse d'une demande de sortie inclut le processus d'analyse d'une demande d'abandon. En effet, pour une demande d'abandon, on vérifie si les exigences d'abandon relatives au rôle, à la mission et au(x) but(s) concernés par la clause sont satisfaites. Tandis que dans le cas d'une demande de sortie, cette vérification est faite pour chaque clause du contrat de l'agent dans l'instance de groupe concernée par la demande de sortie. De plus, dans le cas où l'agent n'est pas membre d'une autre instance de groupe dont les sorties sont gérées par l'instance de portail, la satisfaction des exigences de sortie associées à l'instance de portail est également vérifiée.

L'algorithme d'examen des exigences d'abandon associées à une clause pour la demande d'abandon d'un agent est présenté ci-dessous. Cet algorithme prend en paramètre l'identifiant de l'agent (ag) qui a fait la demande d'abandon et l'identifiant de la clause (cl) concernée par la la demande d'abandon.

– La première instruction permet d'obtenir le contrat de l'agent ag dans l'instance de groupe où est examinée la demande d'abandon.

– Dans la seconde instruction, on suppose que $leaveReqValues$ est une table de hachage dont la clé d'accès à chaque valeur est l'identifiant d'une exigence d'abandon. Cette table est initialisée par toutes les exigences d'abandon de la clause cl. Rappelons que une clause comprend un rôle, une mission et un ensemble de buts (Cf. équation 6.17). D'après les équations 5.3, 5.17 et 5.18 des exigences d'abandon peuvent être associées à chaque *rôle, mission et but*. La fonction $getLeaveRequirement(co.cl)$ de l'algorithme ci-dessous retourne donc l'union des exigences d'abandon du rôle, de la mission et des buts d'une clause cl d'un contrat co.

– La valeur associée à chaque clé de $leaveReqValues$ est un booléen résultat de la fonction $check(req_{id}, ag)$ qui vérifie si l'exigence req_{id} est satisfaite par l'agent ag.

– L'algorithme retourne la table $leaveReqValues$ comme résultat.

Algorithm 2 leaveRequestAnalysis(ag, cl)

```
co = Co.ag
leaveReqValues = getLeaveRequirement(co.cl)
for (req_id ∈ leaveReqValues) do
  leaveReqValues[req_id] = check(req_id, ag)
end for
return leaveReqValues
```

L'algorithme d'examen d'une demande de sortie d'un agent présenté ci-dessous prend en paramètre l'identifiant de l'agent ag et l'identifiant de l'instance de groupe de laquelle l'agent veut sortir. Rappelons qu'il est exécuté dans une instance de portail.

– La première instruction précise que l'algorithme nécessite deux variables globales $clausesReq$ et $exitReqValues$. Elles seront utilisées respectivement pour enregistrer le résultat d'analyse des exigences d'abandon de chaque clause du contrat concerné par la demande de sortie et pour enregistrer les valeurs d'examen des exigences de sortie de l'instance de portail qui gère la demande de sortie.

– $clausesReq$ est donc une table de hachage dont la clé d'accès à chaque valeur

est l'identifiant d'une clause du contrat de l'agent ag, contrat enregistré dans l'instance de groupe igr. La valeur associée à chaque clé est le résultat de l'algorithme 2 ($leaveRequestAnalysis$).

– $exitReqValues$ est une table de hachage dont la clé d'accès à chaque valeur est l'identifiant d'une exigence de sortie définie pour l'instance de portail où est examinée la demande de sortie. Cette table est donc initialisée par toutes les exigences de sortie de l'instance de portail concernée par la demande de sortie. La valeur associée à chaque clé est le résultat de la fonction $check(req_{id}, ag)$.

Algorithm 3 exitRequestAnalysis(ag, igr)

Require: clausesReq, exitReqValues
  clausesReq = getContractClauses(igr, ag)
  for $\big($cl $\in$ clausesReq$\big)$ **do**
    clausesReq = leaveRequestAnalysis(ag, cl)
  end for
  exitReqValues = getExitRequirement()
  for $\big($req$_{id}$ $\in$ exitReqValues$\big)$ **do**
    exitReqValues[req_{id}] = check(req$_{id}$, ag)
  end for

Dans le cas où toutes les exigences d'*abandon* et de *sortie* sont vérifiées, la demande de sortie ou d'abandon est acceptée. Dans le cas contraire, une négociation est engagée entre l'agent qui demande à sortir de l'instance de groupe et l'agent organisationnel en charge de la supervision de l'instance de portail.

6.2.2.2 Négociation, validation et clôture de contrat

L'étape de négociation dans un processus d'abandon de clause ou de sortie d'une instance de groupe intervient généralement lorsque certaines exigences du contrat concerné par la demande ne sont pas satisfaites. La négociation consiste alors en des échanges de messages entre l'agent de domaine et les agents organisationnels qui supervisent d'une part la gestion des sorties au niveau d'un portail où la demande de sortie est traitée d'autre part la gestion des instances de groupe et de schémas sociaux concernées par les clauses litigieuses. Le but de la négociation est d'atteindre un consensus entre les différentes parties afin de permettre à l'agent de domaine d'abandonner la (ou les) clause(s) et éventuellement de sortir de l'instance de groupe et à l'entité organisationnelle de poursuivre son cycle de vie sans trop de difficultés.

En plus de la gestion des entrées et des sorties au sein d'une entité organisationnelle oe, celle-ci doit veiller aux respects des normes qui contraignent le comportement des agents et qui définissent les schémas de coopérations permettant aux agents de réaliser les objectifs collectifs de l'oe. Nous présentons dans la prochaine section notre approche de contrôle.

6.3 Régulation des agents

Une OS définie avec une OML est interprétée à la fois par des agents pour raisonner sur leur comportement au sein d'une entité organisationnelle et également par l'OMI pour gérer la dynamique liée aux actions des agents de domaines dans une OE ainsi que la régulation de leurs comportements. Selon la vision développée dans les organisations normatives, on distingue deux principales approches de contrôle des agents :

– *Régimentation* : elle consiste à mettre en oeuvre des mécanismes qui empêchent à priori toute possibilité pour un agent de pouvoir transgresser certaines contraintes et normes organisationnelles. Par exemple, dans MOISE, les liens de compatibilité et les liens de cardinalités sont gérés dans une entité organisationnelle avec cette approche. Ainsi, il n'est pas possible pour un agent d'adopter un rôle ou une mission, ou un but si la cardinalité maximale de celui-ci est atteinte. Cette approche est également celle utilisée dans les modèles RBAC que nous avons présentés dans le chapitre 2.

– *Régulation* : le principe de cette approche est basé sur le fait que certaines normes d'une organisation définissent les comportements attendus des agents au sein d'une entité organisationnelle. Ainsi, contrairement à la régimentation, des agents d'une OE peuvent transgresser les normes de l'entité organisationnelle. Par exemple, dans MOISE un agent peut s'engager à une mission alors qu'il n'existe pas de clause dans son contrat lui permettant de s'engager à cette mission.

  L'avantage de l'approche de régulation est qu'elle respecte l'autonomie des agents. Cependant leurs comportements peut être contraire à celui attendu et par conséquent rendre incohérent l'entité organisationnelle vis à vis de son OS. Mais, ce type de comportement peut également être favorable à une organisation en fonction des situations. Raison pour laquelle nous avons choisi de l'adopter pour la gestion des normes dans notre approche de gestion

d'organisations ouvertes. Ainsi, pour la mise en oeuvre de la régulation nous distinguons les phases suivantes : (i) *détection* de la transgression de la norme, (ii) la *catégorisation* du caractère néfaste ou non néfaste de la transgression, (iii) *décision* d'application ou pas d'une sanction.

La distinction entre régimentations et régulation conduit à séparer la gestion générale de la régulation au sein d'une entité organisationnelle en : (i) gestion des régimentations définies par l'OS consistant en leur interprétation et vérification, (ii) gestion de la régulation des normes suivant les trois phases énoncées ci-dessus.

6.4 Synthèse

Dans ce chapitre, nous avons présenté les éléments d'une entité organisationnelle à savoir les instances de groupes, schémas sociaux et portails ainsi que notre approche de gestion de l'ouverture. Les instances de groupes, de schémas sociaux et de portails gèrent des fonctionnalités qui mettent en oeuvre les spécifications définies dans l'OS de entité organisationnelle. Elle assurent donc les propriétés d'ouverture à savoir interopérabilité, entrée / sortie et contrôle au sein d'une entité organisationnelle. Les instances de groupes, schémas sociaux et portail sont supervisés par des agents organisationnels (OrgAgent).

En ce qui concerne la gestion du processus d'entrée dans une OE, nous avons proposé trois moyens d'entrer dans une organisation selon le type de la cible associée à une *porte* d'entrée.

Les deux principales étapes d'un processus d'entrée sont : *candidature* et *contractualisation*. Dans la première étape sont gérées : l'appel à candidatures d'agents et l'enregistrement des candidatures, l'examen des candidatures suivie des résultats. Dans la seconde étape sont gérées la création des contrats ainsi que la négociation et validation des clauses de contrat. Les processus d'entrée sont gérés par des instances de portails au sein d'une OE. Les OrgAgent qui supervisent des portails d'entrée ont pour responsabilité de prendre des décisions particulièrement à l'étape de négociation / validation de contrat d'entrée d'un agent. La négociation des engagements d'un contrat dépend du type de porte d'entrée utilisée. Lorsque la porte d'entrée est associée à un $role$, les agents candidats à l'entrée ont la possibilité de négocier les missions auxquelles ils souhaitent s'engager parmi celles qui sont associées au rôle par des normes. Pour une porte d'entrée est associée à une $mission$, les agents candidats à l'entrée ont la possibilité de négocier les buts auxquelles ils souhaitent s'engager

parmi celles qui appartiennent à la mission selon la FS. Par contre, avec une *porte* associé à un but il n'y a pas de possibilité de négociation. Des travaux futurs pourront envisager de négocier certaines propriétés attendus du résultat du but à réaliser.

Nous avons également abordé la mise à jour des contrats des agents. Nous avons vu qu'au sein d'une entité organisationnelle, les contrats des agents ne sont pas statiques. En effet, de nouveaux engagements peuvent être ajoutés à un contrat. D'autre part, il est possible aux agents ayant déjà un contrat dans une OE de d'avoir plusieurs contrats. Nous avons présenté sous quelles conditions et comment est géré la création d'un nouveau contrat pour un agent.

Le fait qu'un agent puisse avoir plusieurs contrats dans une organisation pourrait être considéré comme une limite de notre proposition. Mais, ce choix est justifié par le fait de privilégier une gestion distribuée des groupes, schémas sociaux et portail au sein d'une entité organisationnelle ouverte. Enfin, nous avons présenté le processus de gestion des sorties d'agents d'une OE.

Dans le prochain chapitre, nous présentons l'architecture et les composants notamment les artefacts organisationnels de l'infrastructure de gestion d'organisation (OMI) ORA4MAS que nous avons développé afin de gérer les fonctionnalités de mise en oeuvre des propriétés d'ouverture que sont l'interopérabilité, les entrées / sortie et le contrôle.

Chapitre 7

Infrastructure de gestion d'organisation ouverte : OMI ORA4MAS

Dans le premier chapitre de cette partie, nous avons présenté le langage de modélisation d'organisation Moise que nous utilisons pour spécifier explicitement la structure, les fonctionnalités et les règles qui : définissent une organisation, établissent les contraintes que devront suivre les agents de domaines pour réaliser les objectifs collectifs de leur entité organisationnelle, établissent les procédures qui devront être gérées pour assurer les coopérations des agents ainsi que leur régulation. Nous avons présenté dans le chapitre suivant, les principaux composants d'une entité organisationnelle qui sont obtenus par instanciation d'une spécification organisationnelle définie à partir de MOISE. Une OE comprend donc des agents de domaines, des agents organisationnels et des instances de groupes, de schémas sociaux et de portails. Ces instances maintiennent l'état d'une OE suivant des actions organisationnelles réalisées par les agents de domaines et les agents organisationnels au sein d'une OE.

Dans ce chapitre, nous allons présenter l'infrastructure de gestion d'organisation qui permet la création des instances de groupes, de schémas sociaux et de portails. l'OMI permet également l'exécution des procédures d'entrée et d'intégration d'agents, la gestion des actions de coopération, de coordination, et de régulations ainsi que la gestion des procédures de sorties.

Nous présentons dans un premier temps l'outil technologique de base que nous avons utilisé pour la conception de cette OMI qui est le méta-modèle A&A proposé par A. Ricci et al. [117] avec les concepts d'*agent* et d'*artefact*. Nous présenterons ensuite l'architecture générale de l'OMI, avec la notion d'*artefact organisationnel*, qui

est le concept de base de notre infrastructure. Nous présenterons ensuite les différents artefacts organisationnels définis pour l'interprétation de l'OML Moise et la gestion des instances de groupes, schémas sociaux et portails.

7.1 Fondements de l'infrastructure de gestion

L'OMI ORA4MAS (ORganisational Artifacts and Agents for open Multi-Agent Systems) résulte du constat suivant : dans les SMA, les infrastructures de gestion d'organisation existantes sont essentiellement constituées d'agents généralement appelés *agent manager* qui exécutent toutes les actions qui gèrent la dynamique d'une entité organisationnelle en communiquant par échanges de messages avec des agents de domaines.

Cependant, dans le chapitre 4 nous avons présenté les propriétés des SMA ainsi que celles des agents issus de la littérature. En particulier, les agents sont définis comme étant des entités autonomes, c'est-à-dire qu'ils ont la capacité de décider des actions qu'ils voudraient ou pas réaliser. Dans une organisation multi-agent, cette propriété doit être valable aussi bien pour les agents de domaines que pour les agents managers. Mais, dans les infrastructures de gestion d'organisations existantes, les agents managers semblent être réduits à des entités réactives ou programmes d'exécution d'actions de gestion de la dynamique de leur entité organisationnelle. Cette approche qui est généralement justifiée par une volonté d'utiliser exclusivement des technologies agents ne respecte pas l'autonomie des agents managers de qui dépend le fonctionnement d'une OE.

En effet, des travaux de recherches sur la conception d'environnements multi-agents et la simulation de leurs activités dans leur environnement ont illustré la possibilité de développer des SMA dans lesquels les agents sont plus autonomes. Le principe de ces travaux est basé sur les exemples des environnements humains. Dans ces derniers, de nombreux outils (agendas, téléphone, machine à café, etc.) sont utilisés par des humains afin de les aider dans leurs différentes activités. De la même façon, les environnements des agents peuvent également être conçus de façon à avoir des outils qui aident les agents dans leur interactions et activités. Cette approche a été développé par *A. Ricci et al.* qui ont proposé un méta-modèle appelé *Agent and Artifact*(A&A) [117, 105] de conception et de développement des agents et de leur environnement.

Le concept *artefact* est une abstraction utilisée pour encapsuler les propriétés d'un objet physique ou logique (cf. figure 7.1) qui offre des fonctionnalités que peuvent

utiliser des agents.

Dans la section ci-dessous, nous présentons le principe général ainsi que quelques propriétés du méta-modèle A&A [117, 105]. Ensuite, (section 7.1.2) nous présentons la notion d'artefact avec les éléments qui la définissent.

7.1.1 Caractéristiques du Méta-modèle Agent et Artefact (A&A)

Le principe du méta-modèle A&A est basé sur le fait que les agents dans un SMA ne sont pas des entités isolées dans la nature sans aucune considération de leur environnement. Ainsi, le but du méta-modèle est de fournir des abstractions de représentation des agents et de leur environnement [116]. L'environnement des agents est représenté par la notion de *workspace* [116] : espace de travail. Un workspace est une représentation logique et pas physique de la localisation des agents et des artefacts dans un SMA. En d'autres termes, dire qu'un artefact ou un agent est localisé dans un environnement X revient à dire qu'il est localisé dans le workspace représenté par X.

Après le concept de workspace utilisé pour représenter un environnement SMA, le concept d'artefact a été défini par les auteurs du méta-modèle A&A pour représenter les objets ou ressources d'un environnement multi-agents. Les agents dans le méta-modèle A&A sont considérés comme étant cognitifs, c'est-à-dire capables d'accéder à un environnement (workspace) et d'utiliser les artefacts selon leurs besoins et choix. Cependant, le méta-modèle n'impose aucune architecture d'agent. Les auteurs du méta-modèle ont simplement défini un ensemble de fonctionnalités et primitives que doivent utiliser les agents indépendamment de leur architecture et de leur langage d'implémentation de s'enregistrer dans un workspace et d'utiliser les artefacts.

En plus d'avoir la volonté de fournir une approche conceptuelle de systèmes multi-agents avec une réelle prise en compte et mise en oeuvre de l'environnement, le modèle A&A a également pour objectif d'offrir des outils de développement de SMA ouverts. En effet, en considérant les propriétés d'ouverture étudiée dans la première partie de ce manuscrit, nous verrons dans la section 7.1.2 les moyens qui sont offerts par les artefacts aux agents pour échanger des données avec leur environnement. Ces moyens permettent de répondre partiellement à la propriété d'interopérabilité. En ce qui concerne la propriété d'entrée sortie d'agent dans un environnement et donc dans un workspace, le méta-modèle a défini des fonctionnalités permettant aux agents de s'enregistrer dans un workspace de s'y intégrer en se servant du service de pages jaunes des artefacts pour découvrir les artefacts qu'ils voudraient utiliser. Enfin, le méta-modèle a également défini des fonctionnalités de sortie d'un workspace. Les travaux concernant la mise en oeuvre de la propriété de contrôle sont encore à l'état

embryonnaire. L'idée est d'utiliser l'approche RBAC pour définir des rôles et des permissions d'utilisations des artefacts dans un workspace [103].

La pertinence conceptuelle du méta-modèle A&A avec la représentation d'environnement multi-agent ainsi que les fonctionnalités offertes pour la mise en oeuvre des propriétés d'ouverture sont les caractéristiques qui ont favorisées notre choix de cette approche pour la conception et le développement d'une infrastructure de gestion d'organisations multi-agents ouvertes.

Dans la prochaine section, nous présentons le concept d'artefact dont nous nous sommes inspirés pour définir les briques de base à savoir les *artefacts organisationnels* de notre OMI.

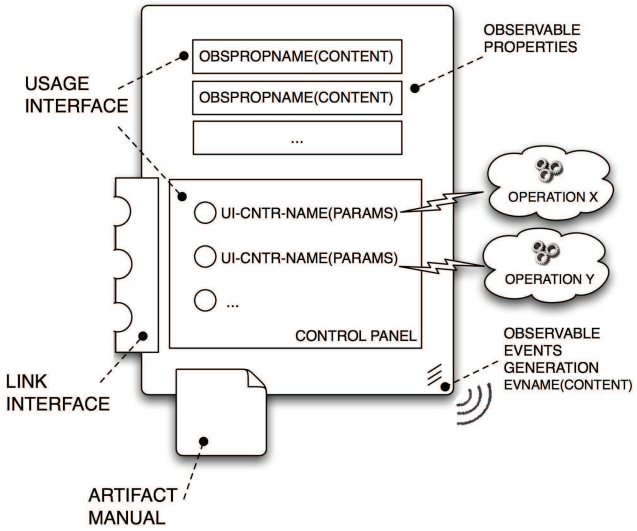


FIGURE 7.1 – Représentation d'un Artefact avec ses principaux éléments : usage interface (UI), link interface (LI) et manuel

7.1.2 Notion d'artefact

Comme nous l'avons dit ci-dessus, le concept d'artefact dans le méta-modèle A&A permet de définir des ressources de nature et de type différents d'un environnement SMA. Ainsi chaque artefact encapsule des fonctionnalités correspondantes à la ressource qu'elle représente. Ces fonctionnalités pourront être utilisées par différents agents dans le workspace ou seront créés les artefacts. D'où la nécessité pour les auteurs du méta-modèle de définir les composants d'un artefact qui permettent

sa manipulation indépendamment des services qu'il fourni. Chaque artefact est donc principalement composé de trois éléments [104, 105]

- Un *manuel d'utilisation* contenant la description des fonctionnalités ou services fournis par un artefact ainsi que les instructions pour l'utiliser.
- Une *interface d'usage* (UI) constituée : (i) de *propriétés observables* (OP) à partir desquelles les agents accèdent aux informations publiques de l'état de l'artefact, (ii) d'*opérations* qui permettent aux agents d'accéder aux services fournis par un artefact.
- Une *interface de lien* (LI) regroupant des *opérations* qui ne peuvent être déclenchées que par d'autre(s) artefact(s). Ces opérations servent à assurer les fonctionnalités de composition de services différents fournis par des artefacts différents. Elles gèrent ainsi les coordinations entre artefacts.

En résumé, seules les opérations et les propriétés observables de l'UI d'un artefact ainsi que son manuel sont accessibles aux agents. Les opérations de la LI ne sont donc pas accessibles aux agents. La figure 7.1 présente les différents composant d'un artefact et la figure 7.2 résume l'architecture du méta-modèle A&A. Dans cette architecture, il est considéré qu'un SMA est constitué d'environnement de travail représenté par des workspaces. Ces derniers sont composés d'artefacts et d'agents. L'architecture montre également les actions qu'un agent peut faire vis-à-vis d'un workspace et qui sont l'entrée (action *joint*) et la sortie (action *quit*). Les actions qu'un agent peut faire sur un artefact sont : la consultation du manuel d'un artefact, l'utilisation de l'interface de contrôle pour l'exécution d'une opération, la perception des événements générés suite à l'exécution d'une opération, et la perception ou l'observation des données fournies par les propriétés observables.

En résumé, un agent utilise un artefact dans un workspace pour accéder au(x) service(s) fourni(s) par celui-ci. L'accès à un service peut se faire par exécution d'une opération de l'artefact. L'exécution d'une opération peut nécessiter des paramètres d'entrées. D'où l'intérêt du *manuel d'utilisation* qui sert à décrire les services fournis par un artefact, les informations représentées par ses propriétés observables et les instructions d'exécution (paramètres d'entrées et événements générés) des opérations qu'offre un artefact. En plus de déclencher des événements, l'exécution d'une opération peut entraîner la mise à jour des données des propriétés observables.

Remarques

Les événements générés par un artefact peuvent être captés par plusieurs agents simultanément. D'autre part, tout agent peut accéder aux valeurs des propriétés ob-

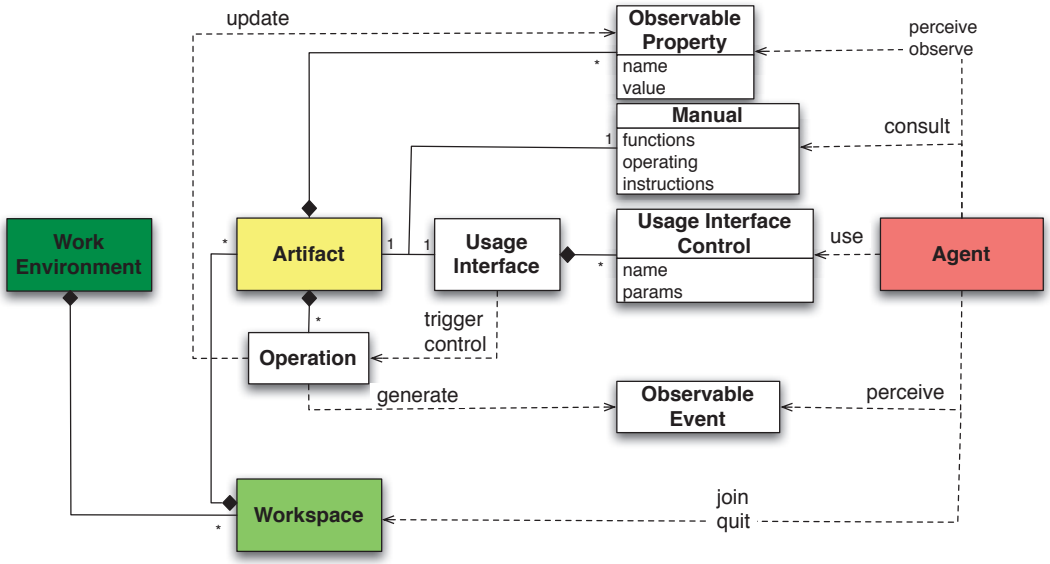


FIGURE 7.2 – Méta-modèle Agent-Artefact

servables de l'UI d'un artefact. Ces deux propriétés sont intéressantes en particulier dans les cas de coopération entre agents qui partagent un ou plusieurs artefacts. En effet, ces propriétés permettent d'assurer la coordination et la synchronisation d'agents dans des activités particulières où ils peuvent avoir besoin des informations de façon synchrone. Exemple : utilisation d'un artefact commun de conception de l'emploi de temps des cours d'une formation par ses différents enseignants pour coordonner l'organisation des enseignements.

L'implémentation du méta-modèle *Agent & Artefact* (A&A) a donné naissance à la plateforme CArtAgO [1] (Common Artifacts and Agents infrastructure for Open multi-agent system) [116]. Cette dernière offre les primitives nécessaire à la création d'environnement de travail d'agents (*worspace*), de manipulation des artefacts (création d'artefacts, appel d'opération, perception des événements générés, perception des informations des propriétés observables, ...). Nous présenterons brièvement cette plateforme dans le chapitre 8 dans lequel nous présentons l'implémentation de l'OMI ORA4MAS dont nous présentons les briques de base dans la prochaine section.

1. CArtAgO : http ://cartago.svn.sourceforge.net/

7.2 Concepts de base et architecture de ORA4MAS

L'infrastructure ORA4MAS a été conçue pour répondre à l'un des objectifs de cette thèse qui est de proposer des outils de modélisation et de gestion d'organisation ouverte. En effet, nous avons vu dans l'état de l'art (cf. chapitre 3) que très peu de modèles organisationnels proposent une approche pertinente pour cette problématique. L'une des raisons de cette limite est liée à l'approche conceptuelle des OMI qui privilégient les échanges de messages comme unique moyen d'interactions entre agents et qui négligent ainsi les avantages que peut offrir l'environnement dans lequel évoluent les agents. L'une des raisons de cette négligence est peut être le manque de technologies adéquates telle que celle proposée par le méta-modèle A&A et offrant une approche conceptuelle intéressante avec des abstractions applicables aux problématiques spécifiques des organisations à savoir la gestion structurelle des agents dans une entité organisationnelle, la gestion de leurs coopérations et coordinations à la réalisation des objectifs collectifs de l'OE et la gestions des normes qui contraignent les comportements des agents.

Ainsi, l'OMI ORA4MAS a été conçu selon l'approche du méta-modèle A&A avec une spécification d'artefacts dédiés à la gestion d'entités organisationnelles dont l'OS a été définie avec le langage de modélisation d'organisation MOISE. Son principe générale est d'être une API dans laquelle des agents de domaines et des agents organisationnels utilisent des artefacts de leur entité organisationnelle –qui représente en quelque sorte leur environnement de travail– pour coopérer à la réalisation des objectifs collectifs de leur OE [76].

Dans le chapitre 6 une entité organisationnelle a été définie comme étant composée d'instances de groupes, de schéma sociaux et de portails d'entrée / sortie. Chacun de ces types d'instances définissent des fonctionnalités qui contribuent à la gestion de la dynamique d'une entité organisationnelle.

L'objectif de l'OMI ORA4MAS est d'offrir les outils technologiques de gestion concrète des fonctionnalités correspondantes aux instances de groupes, de schéma sociaux et de portails ainsi qu'aux activités des agents organisationnels et des agents de domaines. Pour réaliser cet objectif, nous avons défini l'architecture représentée par la figure 7.3. Nous présentons dans les sections qui suivent les principaux concepts de cette architecture.

7.2.1 Artefact organisationnel

Définition 1 : un *artefact organisationnel* (OrgArt) est une entité logicielle qui encapsule des fonctionnalités de gestion d'une entité organisationnelle dont la spécification est définie à partir d'un langage de modélisation d'organisation.

Un OrgArt hérite des caractéristiques d'un artefact telles que définies par le méta-modèle A&A. Les fonctionnalités qu'un OrgArt gère sont : la mise à disposition aux agents d'interface d'interactions avec une entité organisationnelle, la mise à disposition d'informations appropriées pour agents leur qui leur permettent de raisonner sur leurs interactions avec une OE et leur action au sein d'une OE, la gestion de l'exécution d'interactions et la gestion des coordinations avec d'autres artefacts organisationnels pour assurer la modularité et la cohérence au sein de l'organisation.

Un OrgArt est un type d'artefact abstrait dont la réification pour l'OML MOISE a entraînée la définition de types spécifiques d'artefact organisationnel afin d'avoir une homogénéité des fonctionnalités correspondantes aux différents dimensions de MOISE. Ainsi, nous avons défini quatre principaux types d'OrgArt : *GroupBoard* , *SchemeBoard* , *PortalBoard* , *OrgBoard* .

- Un *GroupBoard* assure la gestion d'une instance de groupe suivant sa spécification définie dans la SS.
- Un *SchemeBoard* gère une instance de schéma social dont la spécification est fournie dans la FS ainsi que les normes définies dans la NS qui sont associées aux missions du schéma social.
- Un *PortalBoard* gère une instance de portail.
- Un *OrgBoard* qui gère les informations générales d'une OE. Exemple il sert de service de pages jaunes pour les instances des autres types d'OrgArt (*GroupBoard* , *SchemeBoard* , *PortalBoard*) crées dans une entité organisationnelle.

Puisque les artefacts organisationnels héritent des caractéristiques d'un artefact défini par le méta-modèle A&A, chaque OrgArt comprend une interface d'utilisation (UI) composé des propriétés observables et des opérations, un manuel et une interface de lien (LI). Les propriétés observables d'un OrgArt fournissent une vue de l'état de l'artefact avec les informations qu'elles encapsulent. Les opérations de la UI d'un OrgArt permettent aux agents d'accéder aux différentes actions organisationnelles gérées par l'artefact. Les opérations de liens de la LI permettent aux artefacts organisationnels de se coordonner et ainsi de maintenir la cohérence de la gestion distribuée d'une OE. Nous présentons dans la section 7.3, une description détaillée de chaque type

d'artefact organisationnel.

7.2.2 Agent organisationnel

Définition 2 : un agent organisationnel est un agent qui a des responsabilités de supervision au sein d'une entité organisationnelle. Il est donc chargé de prendre des décisions appropriées au bon fonctionnement d'une OE : décisions de sanctions en cas de violation de contraintes et/ou normes, décisions de réorganisations (modifications de l'OS, des répartitions de rôles et des tâches) pour l'amélioration de la coopération à la réalisation de l'objectif commun de l'organisation.

7.2.3 Architecture conceptuelle de l'infrastructure

L'OMI ORA4MAS est construit sur la base des API CArtAgO et MOISE. La plate forme CArtAgO est une API qui permet la création d'environnement de travail d'agents (workspaces) et des artefacts basés définis selon la spécification du modèle A&A. L'API MOISE est l'interpréteur de l'OML MOISE qui traduit une spécification organisationnelle décrite en XML en objets de façon à ce qu'ils puissent être utilisée par les artefacts organisationnels. A partir des primitives de création et manipulation fournies par la plate-forme CArtAgO, et grâce à l'interpréteur MOISE, ORA4MAS permet de créer des instances des différents types d'OrgArt pour une entité organisationnelle. Ainsi, l'architecture conceptuelle est présentée par la figure 7.3. Les agents organisationnels et de domaine interagissent avec une OE crée avec ORA4MAS en se servant des interfaces d'utilisation des artefacts organisationnels constituées des propriétés observables et des opérations. Afin de permettre à des agents de langages d'implémentation différents tel quel Jason, Jade, Jadex, etc. les développeurs de CArtAgO ont implémenté des frameworks appelés *CArtAgO bridges For Agent Programming Language (CArtAgO-bridge4APL)* qui assurent les médiations entre les primitives des langages agents permettant la gestion des croyances, buts et intentions des agents et les primitives de manipulation des workspaces et des artefacts [115].

En résumé, l'OMI ORA4MAS gère essentiellement la création d'OrgArts d'entités organisationnelles, la gestion de la dynamique des entités organisationnelles : les entrées et les sorties des agents, les coopérations des agents à la réalisation des objectifs collectifs de l'OE et la régulation des agents à travers le contrôle de leurs engagements. Les primitives de création d'artefact organisationnelles, d'appel d'opération, de perception de propriétés observables et d'événements sont ceux définis, implémentés et fournis par la plateforme CARTAGO et les frameworks CArtAgO-bridge4APL.

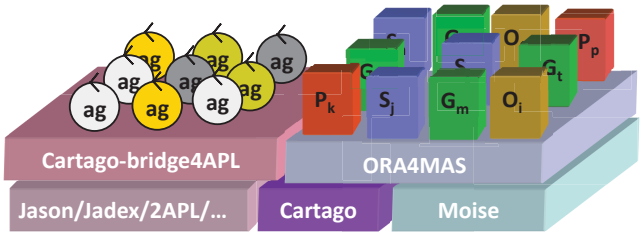


FIGURE 7.3 – Architecture conceptuelle de l'OMI ORA4MAS : Les cercles représentent les agents, les rectangles représentent les différents artefacts organisationnels (GroupBoard (G_i), SchemeBoard (S_j), PortalBoard (P_k), OrgBoard (O_l).

7.3 Artefacts organisationnels pour MOISE

Dans cette section, nous allons présenté l'interface d'utilisation (UI) de chaque type d'artefact organisationnel. La UI d'un OrgArt est constituée des propriétés observables (ObsProp) et des opérations (Op) dont les agents se servent pour interagir avec une entité organisationnelle.

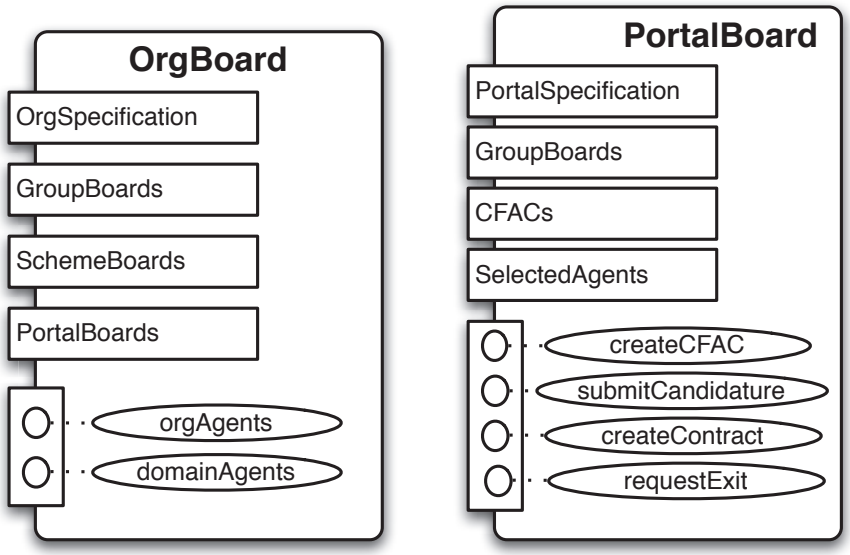
7.3.1 OrgBoard

Un OrgBoard (cf. figure 7.4(a)) peut être considéré comme un registre d'information globale d'une entité organisationnelle.

Il encapsule la spécification organisationnelle de l'OE, et fournit à travers ses propriétés observables les listes des instances de PortalBoard, GroupBoard, SchemeBoard créés dans l'OE. L'intérêt de ces propriétés est qu'elles permettent aux agents pour l'OS d'avoir la description complète de l'entité organisationnelle à partir de laquelle ils peuvent raisonner sur la décomposition structurelle de l'OE, les schémas sociaux les procédures d'entrée/sortie. En ayant également les listes des artefacts organisationnels créés pour la gestion des instances de portail, groupe et schémas sociaux, les agents extérieurs peuvent raisonner sur les portails avec lesquels ils peuvent interagir pour leur procédure d'entrée, les instances de groupes dont ils peuvent être membre, et les instances de schémas sociaux dans lesquelles ils coopéreront avec d'autres agents à la réalisation d'objectifs collectifs de l'OE. En résumé, un OrgBoard est une sorte de vitrine d'une entité organisationnelle.

Propriétés observables d'un OrgBoard

– `OrgSpecification` : la spécification organisationnelle de l'OE.



(a) OrgBoard (b) PortalBoard

FIGURE 7.4 – Artefacts organisationnels pour MOISE : OrgBoad et PortalBoard

– `GroupBoards` : la liste des GroupBoards crées dans l'OE.
– `SchemeBoards` : la liste des SchemeBoards crées dans l'OE.
– `PortalBoard` : la liste des PortalBoards crées dans l'OE.

Opérations d'un OrgBoard

Les opérations de la UI permettent d'obtenir les listes respectives des agents organisationnels `orgAgents` et des agents de domaines `domainAgents` de l'OE.

7.3.2 PortalBoard

Un PortalBoard est un artefact qui assure la gestion d'une instance de portail dans une entité organisationnelle. La figure 7.4(b) montre sa représentation. Une instance de portail étant composée d'un ensemble de portes d'entrée dans des instance de groupes, et d'ensemble d'exigences d'entrée et de sorties (cf. section 6.1.4), les propriétés observables et les opérations d'un PortalBoard servent d'outils permettant aux agents d'obtenir les informations leur permettant de savoir quels appels à candidature créés et de pouvoir y répondre le cas échéant.

Propriétés observables d'un PortalBoard

- `PortalSpecification` : cette propriété observable fournie la spécification du portail d'entrée / sortie. Nous avons vu dans la présentation de la spécification d'entrée / sortie (EES) de l'OML MOISE, (cf. chapitre 5) la description d'un portail. Dans une EES, il est possible de définir plusieurs portails. Pour chaque portail, sont spécifiés des exigences d'entrées et des exigences de sorties, des groupes et des portes. La gestion de chaque instance de portail dépend de sa spécification. La propriété `PortalSpecification` permet pour chaque PortalBoard d'obtenir la spécification de l'instance de portail qu'il gère. Tandis que dans l'OrgBoard, à partir de la propriété observable `OrgSpecification` on obtient la spécification complète de l'OS.

- `GroupBoards` : est la propriété observable qui affiche la liste des **instances de groupes** dont le portail gère les entrées, les sorties ou les deux fonctionnalités. Dans `PortalSpecification` sont spécifiés les types de groupes dont le portail peut gérer soient les entrées, soient les sorties, soient les entrées et les sorties. Les données de cette propriété observable sont présentées sous forme de tableau à deux colonnes dont la première représente l'identifiant d'une instance de groupe et la deuxième la fonction assurée par le portail pour cette instance de groupe. Cette fonction peut être (*Entry, Exit* ou *EntryExit*).

- `CFACs` : propriété observable qui fournit la liste des appels à candidatures publiés et gérés par le portail. Nous rappelons que chaque appel à candidature correspond à une porte qui est associée à une instance de groupe enregistrée dans `GroupBoards`. Nous avons présenté la description d'un appel à candidatures dans la section 6.1.5.

- `SelectedAgents` : cette propriété observable permet d'afficher la liste des agents sélectionnés à l'issus du processus d'analyse de candidatures pour chaque CFAC géré par le portail. Les agents figurant dans la liste de `SelectedAgents` pour un CFAC donnée, pourront poursuivre leur processus d'entrée avec la négociation clauses de leur contrat des engagements relatifs à ce CFAC.

Opérations d'un PortalBoard

- `createCFAC` : opération qui permet à un agent organisationnel de créer un appel à candidature qui sera géré par le PortalBoard.

- `submitCandidature` : opération qui permet aux agents d'enregistrer leurs candidatures pour des CFACs publiés par `CFACs`.
- `createContract` : permet de créer un contrat pour un agent admis dans une instance de groupe géré par un GroupBoard. La création du contrat entraînera la création des clauses correspondantes à la porte de l'appel à candidature suite à laquelle l'agent a été admis.
- `requestExit` : opération qui permet aux agents d'enregistrer leurs demandes de sortie d'une d'instance de groupe.

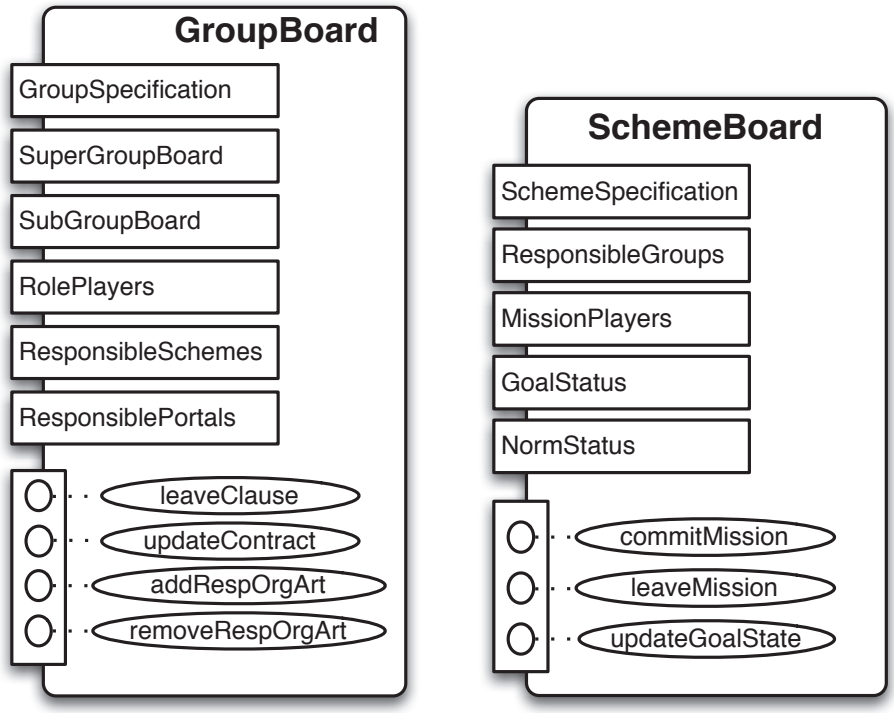
7.3.3 GroupBoard

Le GroupBoard est un type d'artefact qui gère les fonctionnalités d'une instance de groupe. Dans la présentation de l'OML MOISE, nous avons dit que la spécification structurelle d'une OS permet de définir des spécifications de groupes. Un groupe étant défini par un ensemble de rôles, de lien et éventuellement de sous groupes et de groupes dépendants.

les propriétés observables d'un GroupBoard sont (cf. 7.5(a)) :

- `GroupSpecification` : c'est la spécification de l'instance de groupe gérée par le GroupBoard telle qu'elle est définie dans la SS.
- `SuperGroupBoard` : fournit le(s) identifiant(s) de GroupBoard qui gère(nt) chacun une instance de groupe parent de l'instance de groupe de ce GroupBoard.
- `SubGroupBoard` : fournit le(s) identifiant(s) de GroupBoard qui gère(nt) chacun une instance de sous-groupe de l'instance de groupe de ce GroupBoard.
- `RolePlayers` : cette propriété observable fournit la liste des rôles de l'instance de groupe et pour chaque rôle le(s) agent(s) l'ayant adopté(s). Ainsi à travers cette propriété observable, les agents peuvent raisonner sur l'état structurel d'une instance de groupe relativement à ses rôles ainsi qu'à leur cardinalité.
- `ResponsibleScheme` : fournit le(s) identifiant(s) de SchemeBoard qui gère(nt) les instances de schémas sociaux dont est responsable l'instance de groupe géré par GroupBoard.
- `ResponsiblePortal` : fournit le(s) identifiant(s) de PortalBoard qui gère(nt) les entrées et les sorties d'agents de domaine dans l'instance de groupe géré par

le GroupBoard.



(a) GroupBoard (b) SchemeBoard

FIGURE 7.5 – Artefacts organisationnels pour MOISE : GroupBoard et SchemeBoard

Opérations d'un GroupBoard

- `leaveClause` : cette opération permet à un agent de faire une demande d'abandon de clause de son contrat enregistré dans le GroupBoard.
- `updateContract` : cette opération permet à un agent organisationnel de supprimer ou ajouter une clause dans un contrat. Par exemple, si une demande d'abandon de clause faite par un agent avec l'opération `leaveClause` est acceptée, l'OrgAgent utilisera l'opération `updateContract` pour supprimer la clause dans le contrat de l'agent ayant fait la demande d'abandon.
- `addRespOrgArt` : cette opération permet d'une part, d'associer le GroupBoard avec un SchemeBoard. Le GroupBoard devient ainsi responsable de l'instance de schéma social gérée par le SchemeBoard. D'autre part, elle permet d'associer le GroupBoard avec un PortalBoard. Dans ce cas, le PortalBoard gère soient

les entrées d'agents, soient les sorties d'agents, soient les deux fonctionnalités pour le GroupBoard.

– `removeRespOrgArt` : opération inverse de l'opération précédente puisqu'elle permet de supprimer soit un SchemeBoard dont est responsable le GroupBoard, soit un PortalBoard qui est responsable de la gestion des entrées et/ou des sorties d'agents de l'instance de groupe gérée par GroupBoard.

7.3.4 SchemeBoard

Un SchemeBoard est un artefact organisationnel qui gère une instance de schéma social dans une entité organisationnelle. Selon la spécification d'un schéma social présentée dans le chapitre 5, les propriétés observables d'un SchemeBoard sont les suivantes (cf. 7.5(b)).

– `SchemeSpecification` : cette propriété observable fournit la spécification du schéma social. Cette spécification permet aux agents qui la consulte de savoir quelles sont les missions, les buts et les plans du schéma social. Grâce aux plans, les agents peuvent résonner sur leur coordination avec les autres pour la réalisation des buts et missions du schéma social.

– `ResponsibleGroup` : liste des instances de groupe qui sont responsable du schéma social. Cette propriété est permet de savoir quels sont les liens entre une instance de schéma social et les instances de groupes dans une OE.

– `MissionPlayers` : cette propriété observable présente la liste des missions gérées par le SchemeBoard avec pour chaque mission la liste des agents qui se sont engagés à la réaliser.

– `GoalsStatus` : l'ensemble des buts d'un schéma social sont ordonnancés selon des plans qui définissent en quelque sorte les conditions de réalisation de chaque but avec les opérateurs de coordination : séquence, parallèle et choix (cf. figure 5.4). Nous avons vu dans la section 6.1 la fonction `goalStatus` qui définit les changements d'état des buts d'une instance de schéma social et les différents états que peut avoir un but au cours de son cycle de vie. La propriété observable `GoalStatus` présente l'état courant de chaque but d'une instance de schéma social.

– `NormStatus` : cette propriété observable présente pour chaque agent ayant une ou plusieurs clauses de contrat qui concerne(nt) des missions et des buts gérés par le SchemeBoard, l'état des normes relatives à ces clauses de contrat.

Opérations d'un SchemeBoard

– `commitMission` : cette opération permet aux agents ayant un contrat dans un GroupBoard responsable d'un SchemeBoard de s'engager aux missions du schéma social géré par le SchemeBoard. Les missions auxquels l'agent s'engage sont celles qui sont associées aux clauses de son contrat. Dans le cas contraire se sera une infraction ou transgression de norme.

– L'opération `leaveMission(`ag_{id}`, mission`$_{id}$`)` d'un SchemeBoard sera utilisée suite à une procédure de régulation. En effet, lorsqu'un agent s'engagera à une mission qui ne fait pas partie des clauses de son contrat, il en résultera une procédure de régulation au cours de laquelle il pourra être demandé à l'agent en infraction de se désengager de la mission concernée. Cela devra se faire par l'exécution de `leaveMission`.

– `updateGoalState` : opération utilisée par les agents pour notifier la satisfaction ou l'impossibilité de satisfaction d'un but. C'est donc l'opération qui entraîne le changement d'état d'un ou plusieurs but(s) de la propriété `GoalsState` que les agents utilisent pour se coordonner dans la réalisations des buts et missions d'une instance de schéma social.

Ayant présenté chacun des artefacts en lien avec l'OML MOISE, nous illustrerons leur utilisation dans la prochaine section.

7.3.5 Exemple d'utilisation des artefacts organisationnels

La description ci-dessous présente de façon succincte l'utilisation des OrgArts. Une description plus détaillée est présentée dans les chapitres 8 et 9.

Considérons à nouveau l'exemple *write paper* dont nous avons défini la spécification organisationnelle dans le chapitre 5 et quelques éléments d'une entité organisationnelle dans le chapitre 5 (Cf. section 6.2.1).

La création de l'entité organisationnelle $wpoe_1$ correspondante à celle décrite par l'équation 6.18 avec ORA4MAS consiste initialement en la création de ses artefacts organisationnels. Celle-ci est faite par un agent qui sera par défaut un agent organisationnel. Dans la suite nous considérons que l'identifiant de cet agent est OrgAgent_5.

Les artefacts organisationnels de $wpoe_1$ sont les suivants :

- Un OrgBoard avec pour identifiant $wpoe_1$ pour désigner l'oe.
- Un PortalBoard avec pour identifiant $wpipor_1$ pour la gestion de l'instance de portail correspondante.
- Un GroupBoard avec pour identifiant $wpigrp_1$ pour la gestion de l'instance de groupe correspondante.
- Un SchemeBoard avec pour identifiant $wpisch_1$ pour la gestion de l'instance de schéma social correspondante.

Les propriétés observables de chacun de ces OrgArts permettent aux agents d'accéder aux informations de l'entité organisationnelle qui sont gérées par l'OrgArt.
Par exemple :

- `orgSpecification` de l'OrgBoard $wpoe_1$ permet aux agents d'accéder à la spécification organisationnelle complète de $wpoe_1$.
- Par ailleurs, `groupSpecification`, `schemeSpecification` et `portalSpecification` permettent respectivement d'accéder à la spécification de l'instance de groupe gérée par le GroupBoard $wpigrp_1$, le SchemeBoard $wpisch_1$ et le PortalBoard $wpipor_1$.

Les opérations de chaque OrgArt permettent aux agents d'interagir avec l'entité organisationnelle. Exemple :

1. La gestion de l'ajout d'une instance de schéma social dont est responsable une instance de groupe ou d'une instance de portail qui gère les entrées / sorties d'une instance de groupe se fait par l'exécution de l'opération `addRespOrgArt(orgArt`$_{id}$`, fonction)` du GroupBoard.
   Ainsi, pour spécifier le fait que $wpigrp_1$ est responsable de $wpisch_1$, OrgAgent$_5$ exécutera `addRespOrgArt(wpisch`$_1$`, SocialScheme)` sur le GroupBoard $wpigrp_1$. Et pour spécifier que $wpipor_1$ gère les entrées / sorties de $wpigrp_1$, OrgAgent$_5$ exécutera `addRespOrgArt(wpipor`$_1$`, EntryExit)`.

2. La gestion de la suppression d'une instance de schéma social dont est responsable une instance de groupe ou d'une instance de portail qui gère les entrées / sorties de l'instance de groupe se fait par l'exécution de l'opération `removeRespOrgArt(orgArt`$_{id}$`, orgArt`$_{type}$`)` du GroupBoard. Le paramètre orgArt$_{type}$ permet de savoir dans quelle propriété observable (`ResponsibleSchemes` ou `ResponsiblePortals`) orgArt$_{id}$ doit être supprimé.

3. Gestion des entrées
   - La création des appels à candidatures est faite par OrgAgent_5 en exécutant l'opération `createCFAC(` $cfac_{id}$`,` ga_{id}`,` $grpBoard_{id}$`,` $schBoard_{id}$`)` du PortalBoard $wpipor_1$. Les deux premiers paramètre de cette opération sont les identifiants respectifs de l'appel à candidature à créer ($cfac_{id}$) et de la porte correspondante (ga_{id}). L'exécution de cette opération pour les portes ga_1 et ga_2 de `portalSpecification` entraîne l'apparition de deux entrées dans la propriété observable `CFACs` de $wpipor_1$. Ces deux entrées ont pour identifiant respectif $cfac_1$ et $cfac_2$ et pour valeur respective une structure de données représentant toutes les informations d'un CFAC tel que présentées par l'équation 6.5.
   - L'enregistrement d'une candidature se fait par l'exécution de l'opération `submitCanditure(`cfac_{id}`, cand)` qui prend en paramètre l'identifiant de l'appel à candidature correspondant à la candidature et une chaîne de caractères représentant les informations de la candidature telle que présentées par l'équation 6.19.
   - L'opération `createContract(`ag_{id}`,` cfac_{id}`)` permet à un agent organisationnel de créer un contrat pour un agent de domaine ag_{id} dont la candidature a été admise pour un appel à candidature cfac_{id}. La création du contrat entraînera la création des clauses correspondantes à la porte $ga_{cfac_{id}}$ de l'appel à candidature $cfac_{id}$.
     Ainsi pour créer le contrat de l'agent_1 présenté dans la section 6.1.6, l'OrgAgent_5 exécutera l'opération `createContract(`agent_1`,` cfac_1`)`.
   - L'opération `updateContract(`co_{id}`, op, cl` permet selon la valeur de op d'ajouter ou supprimer une clause cl dans un contrat co_{id}.

4. Gestion des engagements aux missions et des coopérations à la réalisation des buts
   - Les agents métiers ou de domaine ayant réussi leur processus d'entrée dans des instances de groupe utiliserons l'opération `commitMission(`ag_{id}`,` mission_{id}`)` du SchemeBoard $wpisch_1$. Les paramètres de cette opération sont l'identifiant de l'agent qui exécute l'opération et celui de la mission à laquelle l'agent s'engage.
   - Lorsque les agents métiers auront à coopérer à la réalisation des buts au SchemeBoard $wpisch_1$ en spécifiant qu'ils ont réalisé ou pas les buts de leur(s) contrat(s), ils devront exécuter l'opération `updateGoalState(`goal_{id}`,`

`goal`$_{state}$`)`. Pour cette opération, `goal`$_{id}$ est l'identifiant du but dont l'agent voudrait modifier l'état avec la valeur `goal`$_{state}$. Exemple, la mise à jour du but $wsec$ de la clause cl_1 du contrat co_1 de l'agent$_1$ à l'état $achieved$, consiste à exécuter `updateGoalState(`$wsec$`,` $achieved$`)`.

5. La gestion des abandons de clauses d'un contrat se ferra par l'exécution de l'opération `leaveClause(ag`$_{id}$`, clause`$_{id}$`)` d'un GroupBoard. Ainsi, si l'agent$_1$ veut abandonner la clause cl_2 de son contrat, il exécutera `leaveClause(ag`$_{id}$`, cl`$_2$`)` du GroupBoard $wpigrp_1$.

6. La gestion des sorties se fait par l'exécution de l'opération `requestExit(ag`$_{id}$`, grpBoard`$_{id}$`)`. Tout agent qui invoque cette opération fournit en paramètre son identifiant et l'identifiant du GroupBoard qui gère l'instance de groupe de laquelle il veut sortir.

7. Les opérations `orgAgent()` et `domainAgent()` de l'OrgBoard ne prennent pas de paramètre. Elles servent à obtenir respectivement la liste des agents organisationnels et la liste des agents de domaines ou métiers de l'entité organisationnelle.

7.4 Synthèse

Dans ce chapitre nous avons présenté l'infrastructure de gestion d'organisations multi-agents ouverte ORA4MAS que nous avons développés. Elle est basée sur les concept du méta-modèle *Agent and Artifact (A&A)* proposé par Ricci et al. Elle s'inspire également des implémentations précédentes de MOISE à savoir $\mathcal{S}$-$\mathcal{M}$OISE$^+$ et $\mathcal{S}$YNAI. Cependant, elle se distingue de ces implémentations par l'utilisation du concept d'*artefact organisationnel* (OrgArt) qui est l'élément de base de cette OMI. En effet, les OrgArts héritent des propriétés des artefacts du méta-modèle A&A et encapsulent les principales fonctionnalités de gestion d'une entité organisationnel : gestion des entrées, coordination des coopérations, gestion des sorties. Ainsi, les agents utilisent les propriétés observables des OrgArts pour accéder aux informations d'une entité organisationnelle et leurs opérations pour réaliser des actions au sein d'une l'entité organisationnelle.

ORA4MAS est constitué de quatre types d'artefacts organisationnels qui permettent de gérer les dimensions d'une organisation représentées avec l'OML MOISE : OrgBoard, PortalBoard, GroupBoard, SchemeBoard.

- L'OrgBoard est un artefact organisationnel qui encapsule et fournit les informations générale d'une entité organisationnelle : spécification organisationnelle, liste des PortalBoard, GroupBoard, SchemeBoard et des agents de domaines ainsi que des agents organisationnels.
- Le PortalBoard gère les entrées et les sorties d'agents au sein d'une entité organisationnelle.
- Le GroupBoard gère les fonctionnalités structurelles d'une entité organisationnelle notamment les contrats dans lesquels sont enregistrés les rôles et engagement de chaque agent dans les groupes.
- Le SchemeBoard gère les aspects fonctionnelles et normatives d'une entité organisationnelle. Elle assure donc la coordination des coopérations entre agent à la réalisation des buts et missions et met à jour les changements d'état des normes associées aux agents relativement aux clauses de leurs contrats.

Une entité organisationnelle spécifiée avec l'OML MOISE et déployée sous ORA4MAS sera donc constituée d'un seul OrgBoard et d'un ou plusieurs PortalBoard, GroupBoard et SchemeBoard.

Afin d'illustrer le fonctionnement de nos propositions de gestion de l'ouverture au sein d'organisations multi-agents présentées dans les chapitres 5, 6 et 7 de cette deuxième partie de mémoire de thèse, nous présentons dans la prochaine partie la mise en oeuvre de ces propositions dans le chapitre 8 avec la description de la programmation de ORA4MAS et l'illustration de son fonctionnement dans le chapitre 9 avec la présentation d'un cas d'étude.

Troisième partie

Mise en oeuvre & Illustration

Chapitre 8

Mise en oeuvre

Dans les chapitres 5, 6 et 7, nous avons présenté respectivement notre proposition de modèle de spécification d'organisations multi-agents ouvertes, notre approche de gestion de l'ouverture et l'architecture générale de la plateforme développée pour la gestion d'organisations multi-agents ouvertes. Dans ce chapitre nous présentons, les principaux éléments de la mise en oeuvre de ces différents modèles. Ceux-ci nous ont permis de mener les expérimentations avec un cas d'étude décrites dans le chapitre suivant. Nous avons structuré la description de l'implémentation de ORA4MAS en trois sections : (1) l'implémentation et l'intégration des extensions de l'API Moise sur laquelle est basée ORA4MAS (Cf. Figure 7.3), (2) l'implémentation des artefacts organisationnels, (3) l'implémentation des agents pour les simulations de gestion d'entités organisationnelles.

8.1 Présentation de l'API MOISE

Dans le chapitre 7, l'infrastructure de gestion d'organisation (OMI) ORA4MAS a été présentée comme étant basée sur l'API MOISE et l'API CArtAgO (Cf. Section 7.2.3). En effet, l'API MOISE peut être considérée comme l'interpréteur en langage de programmation Java d'une spécification organisationnelle (OS) décrite selon le langage de modélisation d'organisation (OML) MOISE dans un fichier XML. Elle est donc constituée de toutes les classes qui permettent de créer et gérer les objets qui représentent l'OS d'une entité organisationnelle.

L'OML MOISE étant une extension de l'OML $\mathcal{M}$OISE$^+$, ORA4MAS s'appuie et étend l'OMI $\mathcal{S}$-$\mathcal{M}$OISE$^+$ développée pour $\mathcal{M}$OISE$^+$. Les différentes versions du code sont disponibles à l'adresse http ://sourceforge.net/projects/moise/files/. L'ex-

tension de la dernière version (*moise-0.8*) a consisté à développer les classes qui implémentent les spécifications de gestion des entrées / sorties. Ainsi, une nouvelle version du schéma-XML correspondant aux spécifications de l'OML MOISE a été définie. Cette nouvelle version est enrichie, principalement, de l'élément *entryExit-specification* qui décrit la structure XML de la dimension entrée / sortie d'une OS. Les éléments *role, mission, goal* de cette version ont également été enrichis avec les éléments *adoptRequirement* et *leaveRequirement*. Ces derniers permettent désormais au concepteur d'une OS de définir les ensembles d'exigences d'adoption et d'abandon des *rôles* d'une SS, des *missions* et des *buts* d'une FS. La version complète du schéma-XML obtenue suite à ces extensions est fournie en annexe A.

L'implémentation des classes correspondant aux nouvelles spécifications sont regroupées dans le package *moise.os.ees* tel que présenté sur la figure 8.1.

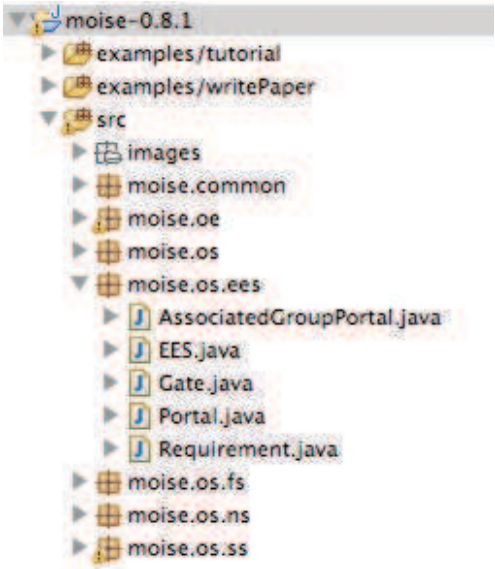


FIGURE 8.1 – Arborescence des packages de l'API MOISE

- La classe **AssociatedGroupPortal** permet de créer les attributs et les méthodes pour la gestion des éléments qui associent un groupe à un portail tel que spécifié dans la section 5.6.1.
- La classe **EES** permet de créer les attributs et les méthodes qui gèrent la liste de tous les portails définis dans la spécification d'entrée / sortie.
- La classe **Gate** permet de créer les attributs et les méthodes qui définissent une porte appartenant à un portail tel que spécifié ans les sections 5.6.1 et 5.6.2.
- La classe **Portal** permet de créer les attributs et les méthodes gérant les ensembles d'exigences d'entrée et d'exigences de sortie, l'ensemble des portes et des groupes associés à un portail (*AssociatedGroupPortal*) selon la spécification de la section 5.6.1.
- La classe **Requiement** permet de créer les attributs et les méthodes qui définissent une exigence conformément à la spécification de la section 5.2.

Nous ne présentons pas les autres packages de l'API car ils sont présentés en détail dans le document [68].

L'ensemble des classes des différents packages de l'API permettent de valider la spécification d'une organisation décrite en XML selon le schéma-XML défini et de créer un ensemble d'objets Java correspondant à une spécification organisationnelle et permettant de régir le comportement des agents dans une entité organisationnelle.

L'ensemble du code de cette nouvelle version de l'API Moise est disponible à l'adresse http ://sourceforge.net/projects/moise/files/.

8.2 Implémentation des artefacts organisationnels

Le développement des artefacts organisationnels de ORA4MAS a consisté à implémenter les mécanismes qui assurent le fonctionnement, la gestion et la régulation d'une entité organisationnelle ouverte dont la spécification est décrite avec MOISE. Ces mécanismes assurent : (1) l'interopérabilité entre les agents et une entité organisationnelle via les propriétés observables et les opérations des artefacts auxquels les agents ont accès, (2) la gestion des coordinations entre les artefacts organisationnelles via les opérations de lien (*link operation*) qui permettent aux artefacts organisationnels de maintenir la cohérence des données (3) et la gestion du cycle de vie d'une entité organisationnelle.

Nous avons présenté dans le chapitre 7, les types d'artefacts organisationnels (OrgBoard, GroupBoard, SchemeBoard, PortalBoard) et pour chaque type ses propriétés observables et ses opérations. Nous présentons ci-dessous les différentes classes mettant en oeuvre ces artefacts, la manière dont sont crées les instances de ces artefacts, leurs propriétés observables et le fonctionnement des opérations principales lorsqu'elles sont exécutées par un agent.

Nous rappelons (Cf. Section 7.1.1) que les artefacts organisationnels héritent des propriétés d'un artefact qui sont développées dans l'API CArtAgo. Ainsi, les mécanismes de base de définition, création et manipulation d'une propriété observable et d'une opération sont ceux fournis par cette API.

Le code java ci-dessous (Listing 8.1) présente la classe abstraite `OrgArt` de laquelle hérite chaque type d'artefact organisationnel. Nous voyons comment sont définies une propriété observable (utilisation de la primitive CArtAgo `defineObsProperty`), une opération (utilisation de l'annotation @OPERATION) et une opération de lien (utilisation de l'annotation @LINK) d'un artefact.

– Les propriétés observables sont créées à partir d'attributs définis dans la classe d'un artefact. Les données d'une propriété observable sont celles de l'attribut correspondant à la propriété observable. Ainsi chaque fois que les valeurs des

données de l'attribut changent la mise à jour de la propriété observable correspondante est faite avec la primitive `updateObsProp`.

– Les opérations sont des méthodes java qui ne retournent pas de données, mais dont l'exécution peut générer des *perceptions* ou *évènement* que les agents peuvent recevoir et interpréter pour connaître le résultat de l'exécution de l'opération. L'émission d'une perception par un artefact est faite avec la primitive `signal`. Une perception est un ensemble de chaîne(s) de caractères séparées par des virgules. Par exemple, l'évènement généré par l'opération `init` dans le Listing 8.1 comprend trois chaînes de caractères : la première identifie l'évènement "Init_OrgArt_ev", la deuxième est le nom de l'artefact organisationnel obtenu par une invocation de la méthode `this.getId().getName()` fournie par CArtAgo et la troisième caractérise le résultat de l'opération "succeeded".

Dans la section 8.3 nous verrons comment les agents captent les évènement qui sont générés par les artefacts.

Listing 8.1 – Code Java de la classe OrgArt

```
package ora4mas.OrgArts;

import alice.cartago.*;
import java.util.ArrayList;
import java.util.List;

/**
 * @author kitio
 */
public abstract class OrgArt extends Artifact{

        private List <String> responsibleOrgAgents = new ArrayList <String>();

        /**
         *  @OPERATION called to initialize an organizational artifact
         *  It defines / creates one observable property with the primitive
         *  defineObsProperty(OBS-PROPERTY-NAME, Correspondent-ATTRIBUTE)
         */
        @OPERATION void init(){

            defineObsProperty("responsibleOrgAgents",responsibleOrgAgents);
        }

        /**
         * @OPERATION called to get the list of all the OrgAgents of the artifact
         * @return the list of all the OrgAgents of an artifact
         */
        @OPERATION List<String> getResponsibleOrgAgents() {
                return responsibleOrgAgents;
        }

        /**
         * @LINK-OPERATION called to add a new agent in the list of OrgAgents
```

```
         * @param agName
         */
        @LINK void addResponsibleOrgAgents(String agName){
                responsibleOrgAgents.add(agName);
                updateObsProperty("responsibleOrgAgents",responsibleOrgAgents);
        }

        /**
         * @LINK–OPERATION called to delete an agent in the list of OrgAgents
         * @param agName
         */
        @LINK void removeResponsibleOrgAgent(String agName){
                responsibleOrgAgents.remove(agName);
                updateObsProperty("responsibleOrgAgents",responsibleOrgAgents);
        }
}
```

8.2.1 Création d'un OrgArt

La création d'un OrgArt se fait en deux étapes : l'initialisation et la configuration.

1. L'**initialisation** permet de créer une instance d'OrgArt. Cependant, les opérations présentées dans le chapitre 7 correspondant au type d'OrgArt de l'instance créée ne sont pas encore exécutables par les agents. En effet, à l'issue de l'initialisation, un OrgArt est dans un état que nous avons nommé *created*. Ses opérations ne seront exécutables que lorsque l'OrgArt sera dans un état *active*.

   Cette démarche résulte de la nécessité de garantir la cohérence des données et le bon fonctionnement des OrgArt d'une entité organisationnelle. Ainsi, des instructions de vérification des valeurs de certaines données ont été définies. Certaines de ces instructions étant complexes (nécessitant des interactions avec d'autres OrgArts) nous les réalisons dans la phase de configuration de l'OrgArt. Par exemple, une instance de GroupBoard / SchemeBoard / PortalBoard n'appartient à une entité organisationnelle que si sa spécification est incluse dans la spécification organisationnelle de cette entité organisationnelle. Cette vérification est faite lors de la *configuration* de l'instance créée au travers d'une interaction (avec une opération lien) avec l'instance d'OrgBoard représentant l'entité organisationnelle.
2. La **configuration** permet d'attribuer des valeurs à certains attributs et propriétés observables d'un OrgArt en exécutant les instructions de vérification de ces valeurs. Si l'opération de configuration d'un artefact organisationnel ne se déroule pas correctement l'OrgArt reste à l'état "created" et aucune de ses opérations ne peut être exécutées par un agent. Un agent pourra soit détruire l'artefact, soit relancer l'opération de configuration avec de nouveaux paramètres.

L'initialisation est une opération qui est exécutée automatiquement lors de la création d'un OrgArt. La configuration, en revanche, est une opération qui doit être invoquée par un agent (nous y reviendrons dans la section 8.3). Selon le type d'un OrgArt, sa configuration peut nécessiter des paramètres. Nous présentons ci-dessous les particularités d'initialisation et de configuration des quatre types d'OrgArt.

Pour illustrer la création des artefacts organisationnels pour une entité organisationnelle avec OR4MAS, nous avons créé les OrgArts de l'exemple "write-Paper" présenté dans le chapitre 5 et dont le fichier XML est dans l'annexe B. Les figures 8.2, 8.3(a), 8.3(b), 8.4(a), 8.4(b), 8.5 présentent les valeurs des propriétés observables des différents OrgArts après leur création (initialisation et configuration).

Initialisation et configuration de l'OrgBoard

Dans le chapitre 7, l'OrgBoard a été présenté comme l'artefact organisationnel dans lequel l'ensemble de la spécification organisationnelle d'une entité organisationnelle oe est mis à la disposition des agents et où sont enregistrés les identifiants de toutes les instances des autres types d'artefacts organisationnels créés au sein de l'oe.

L'initialisation d'un OrgBoard se fait avec le nom du fichier XML de l'OS qui régit la gestion d'une entité organisationnelle. La configuration d'un OrgBoard ne nécessite pas de paramètre. Elle consiste essentiellement en la création de ses propriétés observables. En particulier, la valeur de la propriété observable représentant la spécification organisationnelle est mise à jour. Cette mise à jour est effectuée si la valeur de l'attribut représentant l'OS est différente de nulle, c'est-à-dire si le fichier xml de l'OS a été trouvé et l'objet java représentant la spécification a été correctement créé.

Listing 8.2 – Code Java d'initialisation et de configuration de l'OrgBoard

```
public class OrgBoard extends OrgArt{

private OrgBoardControler myControler;

/**************************************************************************
* This operation initialize the OrgBoard
* @param orgSpecURI is the name of the xml file of the OS
**************************************************************************/
@OPERATION void init(String orgSpecURI) {
        myControler = new OrgBoardControler (orgSpecURI);
        defineObsProperty("orgBoardStatus", OrgArtState.created);
}

@OPERATION void configure() {
 try {
 if (this.myControler.orgBoardSpecification != null) {
```

```
    defineObsProperty(ORGBOARD_OBS_PROP0, this.myControler.orgBoardSpecification.getId());
    defineObsProperty("orgBoardSpecification", this.myControler.orgBoardSpecification);

    signal(INIT_ORGBOARD_EVENT, this.getId().getName(), this.thisOpId.getOpName()+"_op",
                    "succeeded");
   }
   }catch(Exception e) {
    updateObsProperty(ORGBOARD_OBS_PROP5, OrgArtState.error);
    ...
}
     HashMap<String, ArtifactId> groupBoards = new HashMap<String, ArtifactId>();
     defineObsProperty(ORGBOARD_OBS_PROP2, groupBoards);
     ...
     updateObsProperty(ORGBOARD_OBS_PROP5, OrgArtState.active);
     switchToState("active");
   }
          ...
}
```

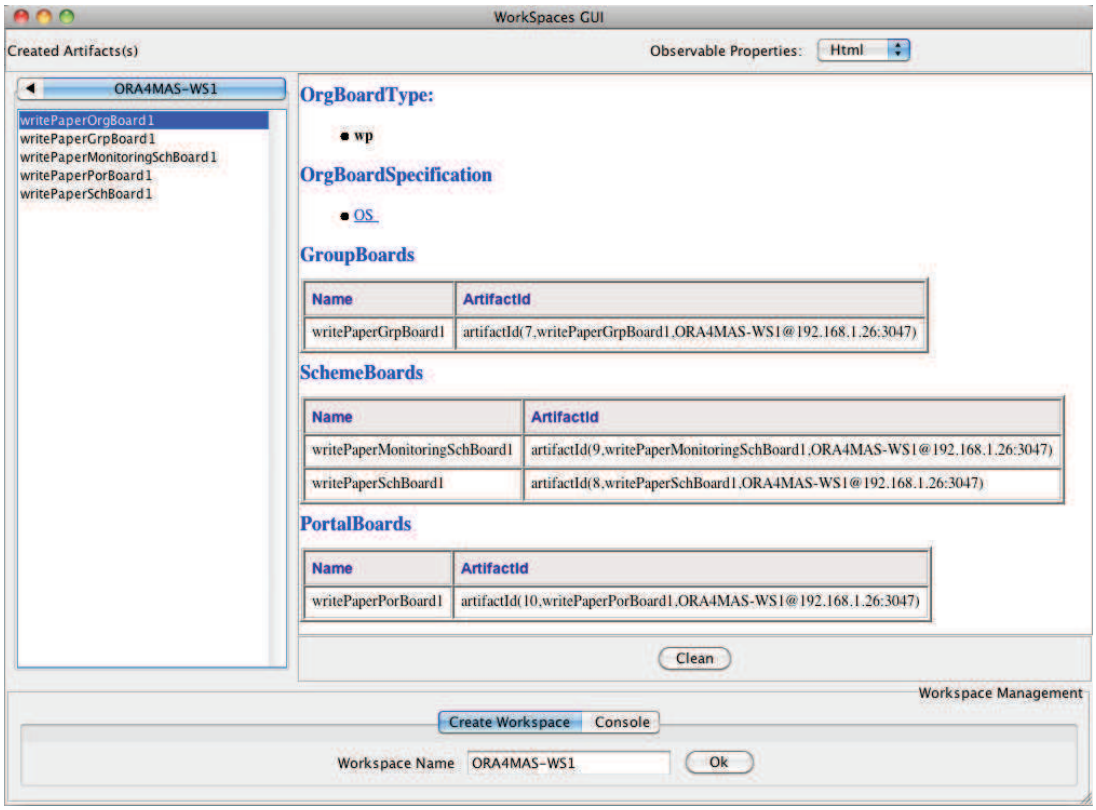


FIGURE 8.2 – Exemple "write-paper" : Propriétés observables de l'OrgBoard après configuraion des autres OrgArts

Initialisation et configuration de GroupBoard, SchemeBoard et PortalBoard

Au cours de l'initialisation de chacun de ces trois types d'OrgArt, quelques propriétés observables sont créées et, pour certaines, leur valeur est initialisée à nulle. La configuration est l'étape la plus importante. Pour faciliter la lisibilité du chapitre, nous ne présentons pas le code de ces méthodes. Il est disponible à l'adresse http ://sourceforge.net/projects/jacamo/files/ora4mas/ où l'API ORA4MAS peut être téléchargée.

La configuration de ces OrgArt (Cf. Listing 8.3) se fait en précisant le nom de l'instance d'OrgBoard (représentant l'entité organisationnelle) à laquelle ils appartiendront, et l'identifiant de la spécification qui régira le fonctionnement de l'OrgArt. Cette spécification correspond respectivement à celle d'un groupe défini dans la SS pour un GroupBoard, à celle d'un schéma social défini dans la FS pour un SchemeBoard et à celle d'un portail défini dans la EES pour un PortalBoard. Le GroupBoard et le SchemeBoard sont configurés en précisant un troisième paramètre.

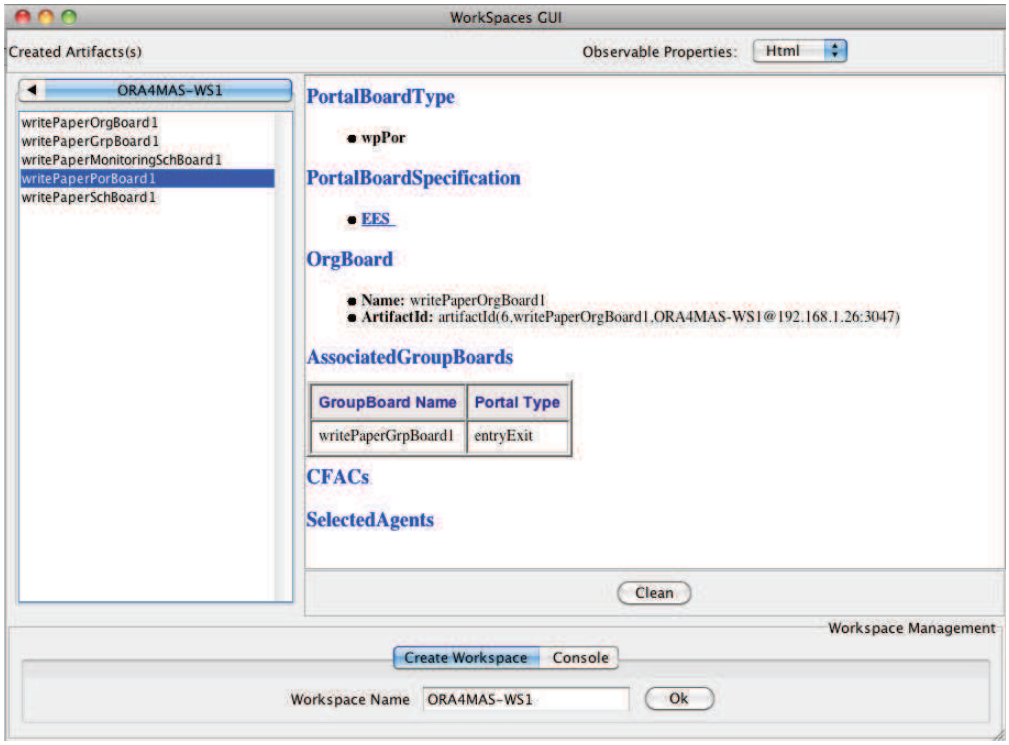
Listing 8.3 – Signatures de *configure* du GroupBoard, SchemeBoard et PortalBoard

```
@OPERATION void configure(Object orgBoardName, Object grpSpecName, Object superGrpBoardName)

@OPERATION void configure(Object orgBoardName, Object schSpecName, Object firstRespGrpBoardName)

@OPERATION void configure(Object orgBoardName, Object porSpecName)
```

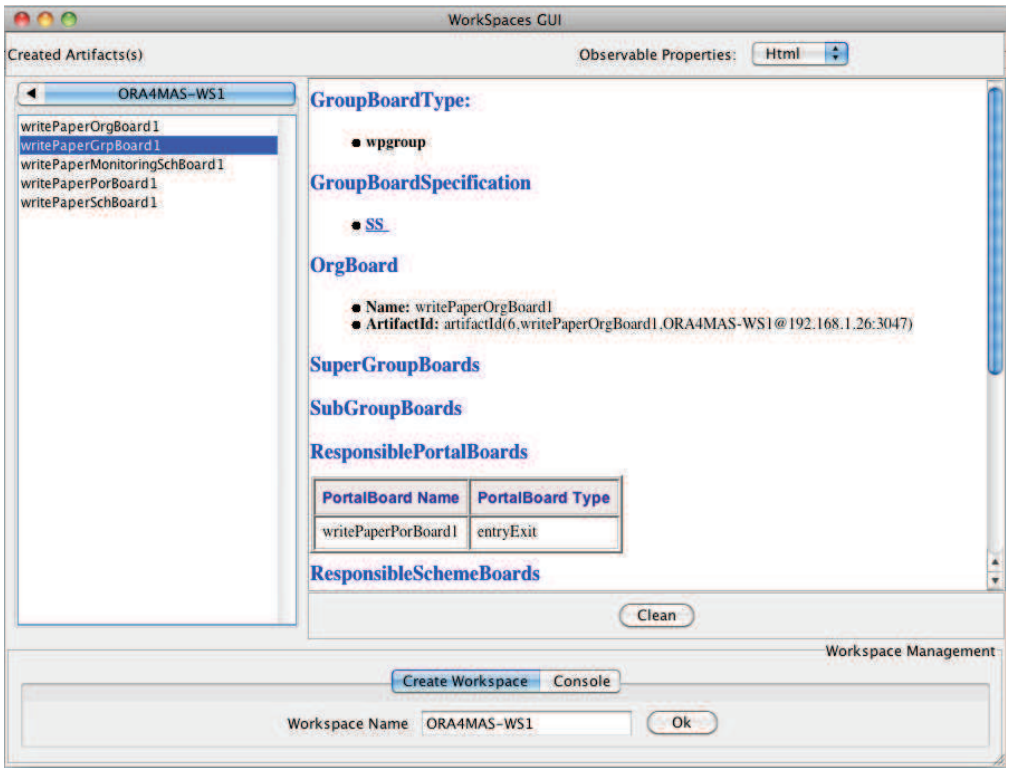
Le troisième paramètre de configuration d'un GroupBoard peut être nul. Dans le cas où la spécification du GroupBoard est celle d'un sous-groupe défini dans la SS, ce paramètre ne doit pas être nul. Sa valeur doit être l'identifiant du GroupBoard parent auquel il devra être associé. Cette valeur sera enregistrée dans la propriété observable `superGroupBoard` (Cf. Section 7.3.3) du GroupBoard à configurer.

A la fin de la configuration l'identifiant du GroupBoard sera transmis au GroupBoard dont l'identifiant a été fourni en paramètre de l'opération de configuration. Il y sera enregistré dans la propriété observable `subGroupBoard` (Cf. Section 7.3.3).

Pour un SchemeBoard le troisième paramètre de configuration ne peut être nul. Il représente l'identifiant d'un GroupBoard qui sera responsable du SchemeBoard. Précisons que d'autres GroupBoard pourront être ajoutés comme responsable du SchemeBoard avec l'opération `addResponsibleOrgArt` du GroupBoard. Rappelons que l'identifiant de chaque GroupBoard responsable d'un SchemeBoard est enregistré dans la propriété observable `responsibleGroups` du SchemeBoard (Cf. Section 7.3.4) et inversement l'identifiant de chaque SchemeBoard dont est responsable un GroupBoard est enregistré dans la propriété observable `responsibleSchemes` (Cf. Section 7.3.3).

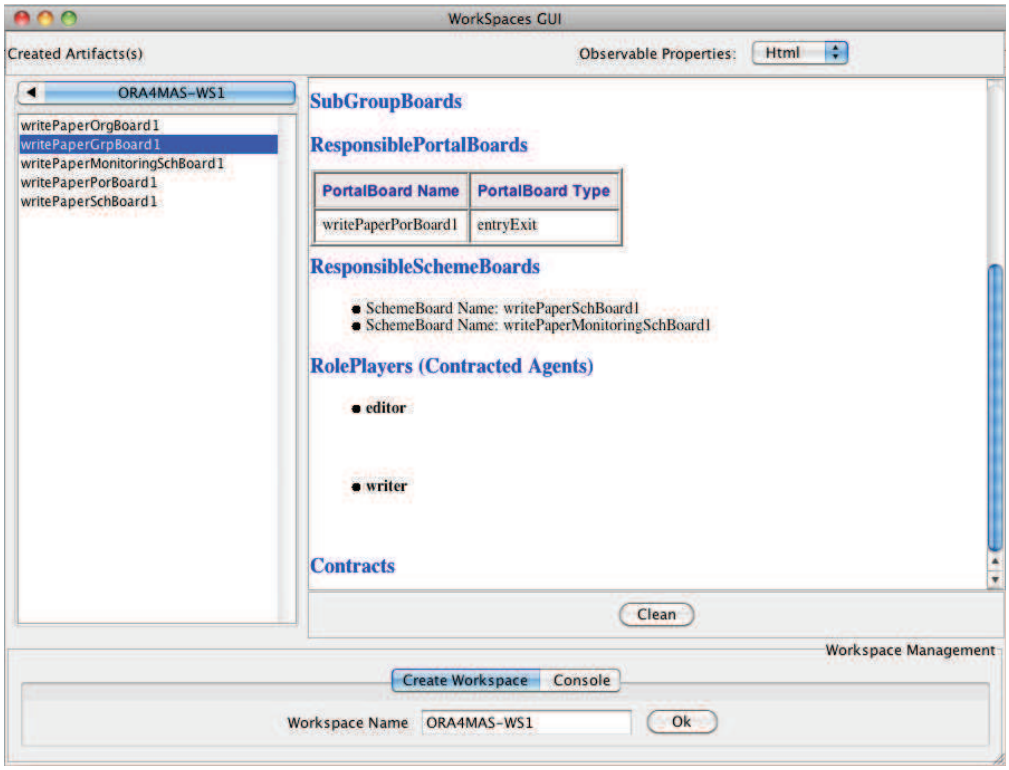


(a) Exemple "write-paper" : Propriétés observables du PortalBoard après configuration

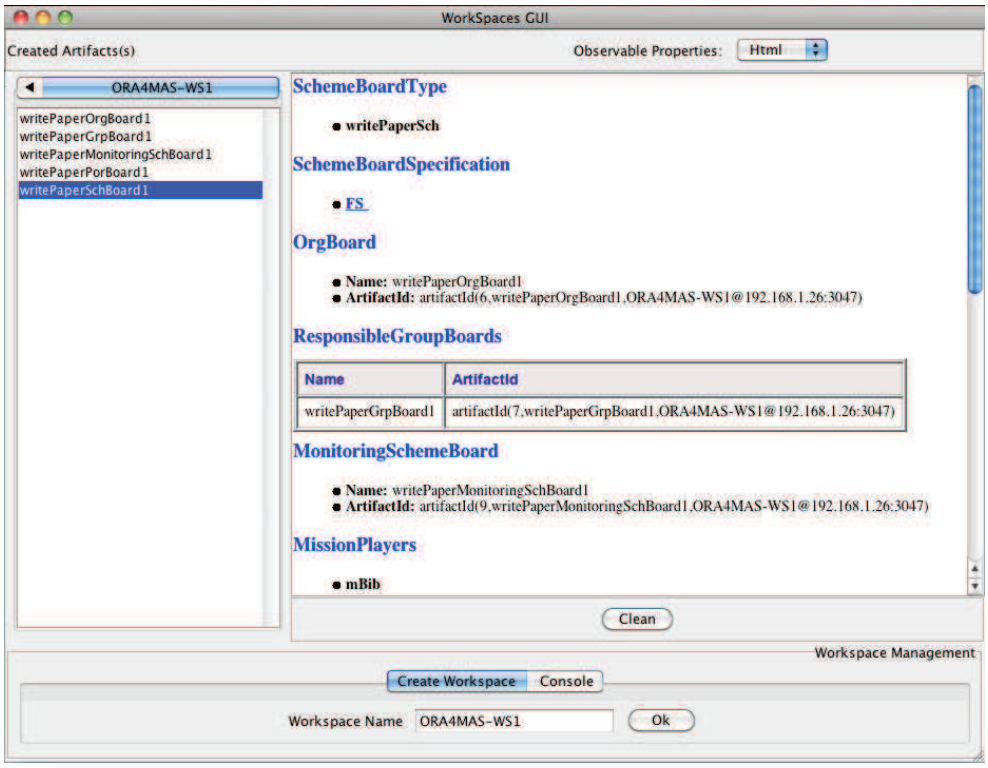


(b) Exemple "write-paper" : Propriétés observables du GroupBoard après configuration (1)

FIGURE 8.3 – Exemple "write-paper" : Propriétés observables du PortalBoard et du GroupBoard après configuration



(a) Exemple "write-paper" : Propriétés observables du GroupBoard après configuration (2)



(b) Propriétés observables du SchemeBoard (1)

FIGURE 8.4 – Exemple "write-paper" : Propriétés observables du GroupBoard et du SchemeBoard après configuration

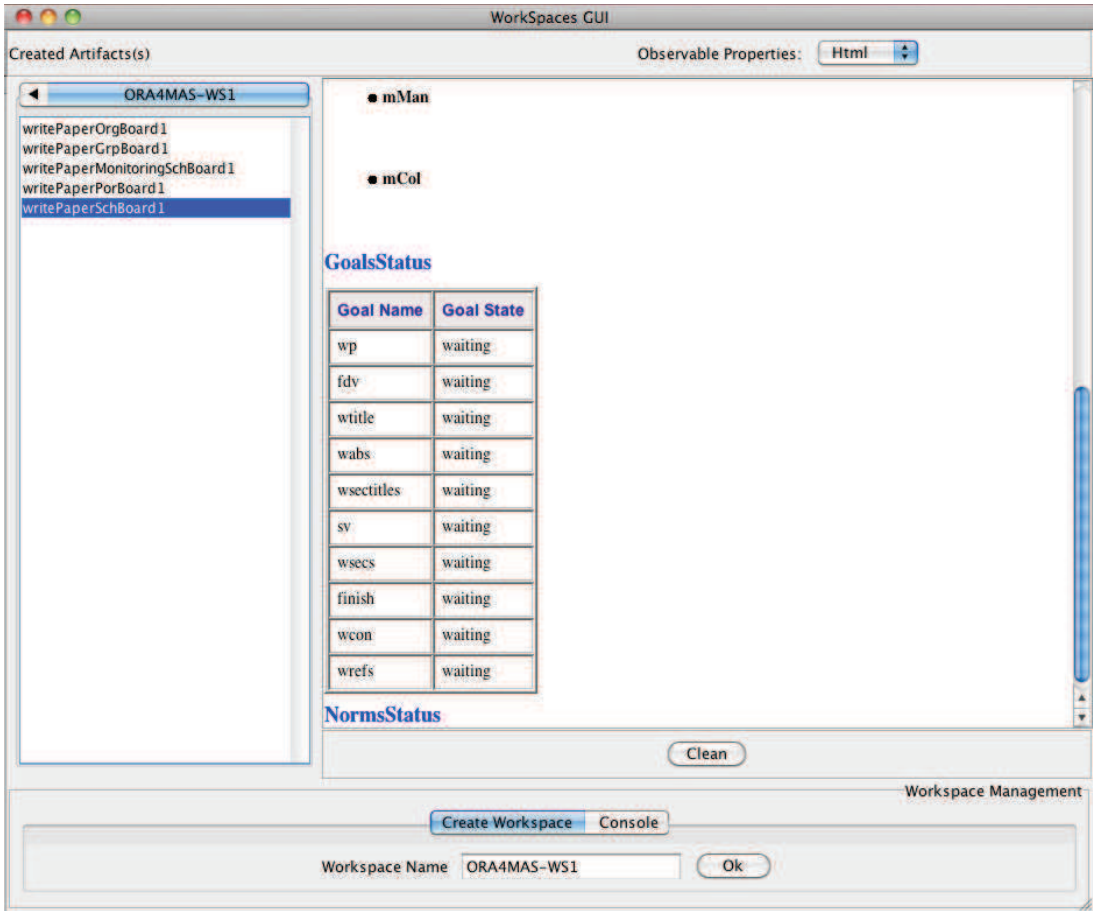


FIGURE 8.5 – Exemple "write-paper" : Propriétés observables du SchemeBoard après configuration (2)

8.2.2 Opérations et propriétés observables des OrgArts

Les opérations et les propriétés observables d'un artefact organisationnel permettent aux agents d'interagir avec l'entité organisationnel à laquelle ils appartiennent.

Les propriétés observables permettent essentiellement aux agents d'obtenir des informations sur l'entité organisationnelle. Comme nous l'avons dit ci-dessus, elle sont définies à partir d'attributs de l'OrgArt auquel ils appartiennent et les valeurs de leur données peuvent changer au cours du cycle de vie de l'entité organisationnelle. Ces changements sont généralement dûs à l'exécution d'une opération. C'est pour cette raison que nous ne présentons pas l'implémentation des propriétés observables dans

une section distincte de celle des opérations. Dans les paragraphes ci-dessous nous présenterons quelques propriétés observables dont les valeurs des données changent suite à l'exécution des opérations dont nous décrivons l'implémentation.

Les opérations encapsulent les mécanismes de gestion des entrées / sorties, des engagements aux missions et de régulation des agents. Elles ne peuvent être exécutées que si l'OrgArt est dans l'état "*active*". Nous présentons ci-dessous deux opérations de chacun des type d'OrgArt PortalBoard, GroupBoard et SchemeBoard. Nous ne présentons que deux opérations par type d'OrgArt pour réduire la longueur du chapitre. Les opérations que nous avons choisis de présenter sont celles qui assurent les principales fonctionnalités du type d'OrgArt auquel elles appartiennent.

Opération du PortalBoard

1. **createCFAC** : cette opération sert à créer des appels à candidatures qui seront gérés par le PortalBoard. La signature de cette opération est présentée ci-dessous.

```
@OPERATION void createCFAC(String cfacId, String gateId, String grpBoardId,
                String schBoardId)
```

   L'exécution de cette opération a besoin de quatre paramètres :

   – *cfacId* : l'identifiant qui sera attribué à l'appel à candidature.

   – *gateId* : l'identifiant de la porte définie dans la spécification de l'instance de portail gérée par le PortalBoard. C'est cette porte qui fournira les principales données (cible, exigences, ...) du CFAC (Cf. Section 6.1.5).

   – *grpBoardId* : l'identifiant de l'instance de GroupBoard dans laquelle est géré le rôle associé à la cible de *gateId*. C'est dans cette instance de GroupBoard que seront enregistrés les candidats qui seront retenus pour *cfacId*.

   – *schBoardId* : la valeur de ce paramètre peut être nulle si la cible de *gateId* est de la forme $(r, \_, \_)$. Dans le cas contraire sa valeur est l'identifiant de l'instance de SchemeBoard dans laquelle est gérée la mission associée à la cible de *gateId*. C'est dans cette instance de SchemeBoard que devront s'engager à la mission concernée, les agents dont la candidature aura été retenue pour *cfacId*.

   L'exécution de cette opération consiste à créer un objet CFAC et à activer le compteur de sa durée de publication ($timeout$ Cf. Section 6.1.5). L'appel à candidature est alors enregistré dans la propriété observable `CFACs` du PortalBoard.

Un exemple d'appel de cette opération par un agent pour les porte `ga1` et `ga2` de l'exemple write-paper est présenté dans l'annexe 6.1.1.

Les figures 8.6(a) et 8.6 présentent les données de la propriété observable `CFACs` dans un exemple d'instance de PortalBoard.

2. **submitCandidature** : la signature de cette opération est la suivante.

```
@OPERATION void submitCandidature(String cfacId, String candidature)
```

Ses deux paramètres représentent respectivement (1) l'identifiant de l'appel à candidature pour lequel l'agent soumet sa candidature et (2) une chaîne de caractères qui comprend toutes les informations d'une candidature tel que décrit dans la Section 6.2.1.1.

Son exécution entraîne la création d'un objet candidature après une analyse syntaxique du deuxième paramètre afin d'en extraire des données qui seront enregistrées dans la candidature. L'objet créé est ensuite enregistré dans une liste de candidatures de l'appel à candidature $cfacId$. Les candidatures sont analysées à la fin du délai de publication ($timeout$) du CFAC si sa valeur est finie, dans le cas contraire, elle sont analysées au fur et à mesure de leur soumission. Le résultat de l'analyse des candidatures est enregistré dans la propriété observable `selectedAgent` dont un exemple est présenté sur la figure 8.7(a).

Lorsqu'un agent est admis pour un appel à candidature et que son contrat est créé et validé, il est transmis au GroupBoard associé à l'appel à candidature pour y être enregistré. Les propriétés observables `RolePlayers` et `Contracts` sont mises à jour. Les figures Fig. 8.7(b), Fig. 8.8(a) et Fig. 8.8(b) montrent des exemples de valeurs des ces propriétés observables.

Opération du GroupBoard

1. **addRespOrgArt** : la signature de cette opération est présentée ci-dessous. Elle permet d'associer à une instance de GroupBoard soit des instances de PortalBoard qui gèreront les entrées et / ou les sorties des agents dans l'instance de GroupBoard, soit des instances de SchemeBoard dont les agents ayant un contrat dans le GroupBoard, pourront être en charge.

```
@OPERATION void addRespOrgArt(String orgArtId, String OrgArtType)
```

Le premier paramètre de l'opération est l'identifiant de l'artefact organisationnel à enregistrer et le second paramètre est son type. Ce dernier permet de savoir dans quelle liste l'OrgArt doit être enregistré. Lorsque l'enregistrement se déroule correctement, le résultat est visible selon le type de l'OrgArt par les propriétés observables `responsiblePortals` et `responsibleSchemes` du GroupBoard (Cf. Figure 8.3(b)) et dans la propriété observable `GroupBoards` du PortalBoard si le type de l'OrgArt est GroupBoard ou `responsibleGroups` du SchemeBoard si le type est SchemeBoard (Cf. Figure 8.3(a) et 8.3(b)).

2. **leaveClause** : cette opération permet aux agents de faire une demande d'abandon de clause de leur contrat après que celui-ci soit enregistré dans le GroupBoard. Une demande d'abandon de clause implique l'analyse des exigences d'abandon associées aux rôle, mission, et buts qui définissent la clause. Elle peut donner lieu à un échange de messages entre l'agent qui souhaite abandonner la clause et l'agent organisationnel qui supervise le GroupBoard. En cas d'accord entre l'agent organisationnel qui supervise le GroupBoard et l'agent qui veut abandonner la clause, cette dernière est supprimée dans le contrat. La signature de cette opération est la suivante.

```
@OPERATION void leaveClause(String agentId, String clauseId)
```

L'exécution de l'opération supprime la clause dont l'identifiant est $clauseId$ dans le contrat de l'agent dont l'identifiant est $agentId$.

Opération du SchemeBoard

1. **commitMission** : cette opération permet aux agents de s'engager explicitement dans une instance de SchemeBoard aux missions pour lesquelles ils ont une ou plusieurs clause(s) dans leur(s) contrat(s) (CF. section 7.3.4). La signature de l'opération est présentée ci-dessous.

```
@OPERATION void commitMission(String agentId, String missionId)
```

Cette opération nécessite deux paramètres : l'identifiant de l'agent qui veut s'engager et l'identifiant de la mission à laquelle il veux s'engager. Avant l'exécution de cette opération, l'état de la mission $missionId$ pour l'agent $agentId$ dans la propriété observable `NormStatus` est *notCommited* dans l'instance de SchemeBoard concerné par l'opération (Cf. Figure 8.11(a)). Lorsque l'exécution de l'opération se déroule correctement l'état de la mission est mis à

jour avec la valeur $commited$ dans la propriété observable `NormStatus` et le nom de l'agent est rajouté dans la liste des nom des agent qui se sont engagés à la *missionId* dans la propriété observable `MissionPlayers` (Cf. Figure 8.9(a)).

2. **updateGoalState** : cette opération présentée comme la précédente dans la Section 7.3.4 permet aux agents de mettre à jour l'état des buts qu'ils ont à réaliser. La signature de l'opération présentée ci-dessous montre qu'elle nécessite deux paramètres. Le premier est l'identifiant du but dont l'agent voudrait mettre à jour l'état et le second est le nouvel état du but.

```
@OPERATION void updateGoalState(String goalId, String goalState)
```

Pour que cette opération s'exécute correctement pour un but dont l'identifiant est $goalId$, l'état de ce but doit avoir la valeur $active$ dans la propriété observable `goalStatus`. Dans le cas contraire l'exécution de l'opération génère une erreur. Les valeurs de *goalState* peuvent être $achieved$ ou $impossible$. Dans le cas où l'exécution se déroule correctement c'est-à-dire sans générer d'erreur, l'état du but est mis à jour avec sa nouvelle valeur.

Un exemple de changement des valeurs des buts de la propriété observable `goalstatus` est présenté par les figures Fig. 8.11(a) et Fig. 8.11(b).

8.3 Agents utilisant ORA4MAS

Dans le chapitre 3 les systèmes multi-agents ont été définis comme des systèmes dans lesquels des agents hétérogènes c'est-à-dire d'architectures différentes (BDI, ...), de langages de programmation différents (Jade, Jadex, Jason, 2APL, ...) peuvent entrer et réaliser leurs actions. Ainsi, les auteurs de CArtAgo ont développé des "ponts" entre CArtAgo et différents langages de programmation d'agents (Jadex, Jason, 2APL) de façon à permettre aux agents développés avec ces langages de pouvoir utiliser les artefacts sur la plateforme CArtAgo. Ces "ponts" sont représentés sur la figure 7.3 par la couche *Cartago-Bridge*.

Les organisations multi-agents étant un sous-domaine des SMA, ceux qui sont ouverts ou veulent l'être doivent offrir cette propriété. L'infrastructure de gestion d'organisation (OMI) ouverte ORA4MAS étant basée sur l'API CArtAgo, elle bénéficie des travaux de cette API pour garantir l'hétérogénéité des agents qui interagissent avec les OrgArts des entités organisationnelles qui y seront créées.

Nous avons développé des agents avec le langage Jason pour réaliser nos expérimentations de gestion d'entités organisationnelles ouvertes. Ceux-ci peuvent créer des artefacts organisationnels, exécuter leurs opérations aussi bien pour les procédures d'entrée / sortie que pour les coopérations à la réalisation des buts des schémas sociaux. Ils se servent des primitives fournies par l'API CArtAgo présentées dans le tableau 8.1 pour la manipulation des artefacts.

Les primitives de CArtAgo permettant de manipuler les artefacts sont regroupées en trois catégories selon leurs fonctions : *entrée / sortie dans un worksapce*, *utilisation des artefacts*, *perception de l'environnement* [105]. En effet, un workspace est un concept défini dans le méta-modèle A&A [117] pour représenter l'environnement d'un SMA. Dans A&A, les artefacts sont créés dans des workspaces et les agents manipulent les artefacts dans des workspaces. Ainsi, les primitives de la première catégorie `joinWorkspace` et `quitWorkspace` permettent aux agents d'entrer et de sortir d'un workspace. La deuxième catégorie des primitives `makeArtifact`, `use`, `lookupArtifact`, `observeProperty` et `disposeArtifact` permettent respectivement de créer un artefact, d'exécuter une opération, de rechercher un artefact dans un workspace, d'obtenir les données d'une propriété observable et de supprimer un artefact. Les primitives de la dernière catégorie `focus`, `sense`, `stopFocus` permettent respectivement de percevoir tous les événements générés par un artefact, de percevoir un seul évènement à un moment donné, et d'arrêter de percevoir tous les évènements d'un artefact.

Les codes des agents présentés à l'annexe C sont ceux de quelques agents que nous avons développés pour réaliser nos expérimentations de création et la gestion des artefacts organisationnels d'un exemple d'entité organisationnelle dont l'OS est write-paper.

A travers ces codes, nous pouvons constater que les agents ne s'envoient pas des messages en permanence afin de se coordonner pour la réalisation des buts du schéma social. Cette possibilité est offerte grâce à la primitive `focus` sur l'artefact organisationnel *writePaperSchBoard*. La coordination des agents se fait par la perception de l'évènement qui leur annonce que l'état des buts du schéma social qu'ils ont à réaliser est *possible* et qu'ils peuvent donc entreprendre leur réalisation et mettre à jour l'état les buts concernés à *achieved* lorsqu'ils auront achevé de les réaliser. De même, les argents organisationnels se servent de cette primitive pour percevoir les évènements générés par les artefacts organisationnels, ainsi que de la primitive `observeProperty` pour percevoir le fonctionnement de leur entité organisationnelle et les comportements des agents afin d'assurer leur fonction de régulation.

8.3.1 Régulation des agents

Dans la section 6.3 nous avons énoncé notre démarche de régulation des comportements malveillants des agents en trois phases : (i) *détection* de la transgression d'une norme, (ii) la *catégorisation* du caractère néfaste ou non néfaste de la transgression, (iii) *décision* d'application ou pas d'une sanction.

- La phase de **détection** est faite par les artefacts organisationnels. En effet, un agent membre d'une instance de groupe géré par un GroupBoard qui est responsable d'un SchemeBoard peut s'engager à une mission de ce SchemeBoard alors qu'il n'existe pas de clause dans son contrat impliquant la mission. Ce comportement étant une transgression de norme, le SchemeBoard ayant exécuté l'opération génèrera une perception –ou évènement– qui signale la transgression. De plus, l'identifiant ou nom de l'agent apparaîtra alors dans la propriété observable `MissionPlayers`.
- La phase de **catégorisation** de la transgression de norme en *néfaste* ou *non néfaste* pour l'entité organisationnelle est faite par l'agent organisationnel en charge de la supervision du SchemeBoard.
- La phase de **décision** est également faite par l'agent organisationnel.

Exemple : nous avons simulé une gestion d'entité organisationnelle ouverte avec transgression de norme pour l'exemple "write-paper". Pour cette simulation, nous avons développé un agent organisationnel *tom* trois agents de domaine *alice*, *jeanne* et *bob*. Le code de ces agents est fourni dans les annexes C.1, C.2, C.3, C.4.

- L'agent organisationnel *tom* créé les artefacts organisationnels de l'entité organisationnel ainsi que les appels à candidatures *cfac1* et *cfac2* qui seront gérés par le PortalBoard (Cf. Figure 8.6). Ensuite il supervise les procédures d'entrée et les actions des agents admis dans l'organisation.
- **Processus d'entrée**
  - L'agent *bob* soumet une candidature pour *cfac2* dont la cible de la porte est (editor, mMan, _). Il est admis pour cet appel à candidature (Cf. figure 8.7(a)). Son contrat a été créé, validé et enregistré dans le GroupBoard (Cf. figure 8.8(a)) et son nom est enregistré dans la liste des agents qui jouent le rôle `editor` dans le GroupBoard (Cf. 8.7(b)).
  - Les agents *jeanne* et *alice* soumettent des candidatures pour *cfac1* dont la cible de la porte est (writer, _, _). Ils sont admis et leurs contrats ont été

créés, validés et enregistrés dans le GroupBoard (voir figures 8.7(a), 8.7(b) 8.8(a), 8.8(b)).

- **Engagement aux missions**
  - L'agent *alice* s'est engagé aux missions *mCol* et *mBib* (Cf. figure 8.9(a)).
  - L'agent *jeanne* s'est engagé sur la mission *mMan* bien qu'il n'existe pas de clause dans son contrat relative à cette mission. Voir figure 8.9(a), 8.8(b) 8.9(b).
  - L'agent *bob* ne peut pas s'engager à la mission *mMan* car la cardinalité maximale de celle-ci est "1" et l'agent *jeanne* s'est déjà engagé à cette mission.
- **Traitement de la transgression de norme** : (Cf. Annexe C.1, C.4).
  L'agent organisationnel *tom* reçoit l'évènement signalant la transgression de norme et le catégorise comme néfaste pour l'entité organisationnelle puisque l'agent tom a été admis pour réaliser cette mission sur la base d'une analyse d'exigences. Il décide donc d'envoyer un message à l'agent jeanne pour lui demander d'abandonner la mission *mMan* et se met à l'écoute de l'évènement généré par une exécution de l'opération `leaveMission` du SchemeBoard par l'agent *jeanne*. Si au terme d'un délai l'agent *jeanne* n'abandonne pas *mMan*, *tom* peut prendre la décision de sanctionner *jeanne*.
  Dans notre simulation, l'agent jeanne a exécuté l'opération `leaveMission` aussitôt après avoir reçu le message. L'exécution de celle-ci a supprimé son nom dans la propriété observable `MissionPlayers`. L'agent bob s'est engagé à la mission pour laquelle il a été admis dans l'entité organisationnelle (Cf. Fig. 8.10(a), 8.10(b)).

  La stratégie de décision appliquée dans ce cas est considérée comme une stratégie d'avertissement. Selon les applications et les cas de transgression de norme, on pourra avoir des stratégies différentes : sanction immédiate, avertissement puis sanction, pas de sanction ...
- **Exécution coordonnée des buts du SchemeBoard**
  Tant que la cardinalité minimale de chaque mission du SchemeBoard n'est pas atteinte, l'exécution des buts du schéma ne peut pas commencer. Tous les buts sont mis en attente `-waiting-` (cf. Fig. 8.5). Après la régulation de la transgression de norme ci-dessus, chaque agent s'est engagé au(s) mission(s) de son contrat et la cardinalité minimale de chaque mission est atteinte. L'état des premiers buts est automatiquement mis à `possible` (cf. Fig. 8.10(a)). L'exécution

du plan du SchemeBoard peut commencer. L'état d'un but racine de plan dans un schéma social est automatiquement mis à `achieved` –en fonction de l'opérateur de plan (séquence, choix ou parallélisme) – lorsque ses sous-buts sont exécutés (e.g. Fig. 8.11(a)).
Tout au long de l'exécution du plan, les agents sont informés des changements d'état des buts en observant `goalsStatus` du SchemeBoard. Les agents déclarent qu'un but a été réalisé par l'opération `setGoalState`. Avant de s'exécuter, cette opération vérifie que l'état du but est `possible`, que l'agent est engagé sur la mission associée à ce but. Si ces conditions sont vérifiées, un évènement annonçant le succès de l'opération est généré et l'état du but est mis à jour dans la propriété observable `goalsState` du SchemeBoard avec la valeur `achieved` ainsi que dans la propriété observable `normStatus` du SchemeBoard pour l'agent ayant exécuté l'opération `setGoalState`. Si des buts sont encore dans l'état `waiting`, l'état du (ou des) but(s) suivant(s) est (sont) mis à `possible` en fonction de l'opérateur du plan, sinon le but racine du plan est automatiquement mis à `achieved` (Cf. Figure 8.11(b)).

8.4 Synthèse

Dans ce chapitre nous avons décrit la mise en oeuvre de l'infrastructure de gestion d'organisation multi-agent ouverte (OMI) ORA4MAS. Celle-ci est basée sur l'API moise qui permet de créer une spécification organisationnelle en Java qui a été définie avec l'OML MOISE et qui est décrite dans un fichier XML. Nous avons donc montré quelles sont les extensions qui ont été apportées à cette API afin de lui permettre d'intégrer nos propositions de spécification de processus d'entrées / sorties.

Nous avons ensuite présenté comment sont développés les artefacts organisationnels de ORA4MAS. Nous rappelons qu'ils sont basés sur l'API CArtAgo qui implémente le méta-modèle Artefact & Agent et donc les spécificités des artefacts qui favorisent le développement d'application multi-agent ouvertes. Nous avons principalement montré comment sont créés les quatre types d'artefacts organisationnels (OrgBoard, PortalBoard, GroupBoard, SchemeBoard) de ORA4MAS. Nous avons ensuite présenté quelques opérations des OrgArts qui permettent aux agents d'interagir avec une entité organisationnelle notamment pour soumettre une candidature d'entrée, pour supprimer une clause de contrat, pour s'engager à une mission et pour coopérer à la réalisation des buts d'un schéma social. La présentation de ces opérations a été illustrée par des exemples de résultat de leur exécution avec des copies d'écrans d'ex-

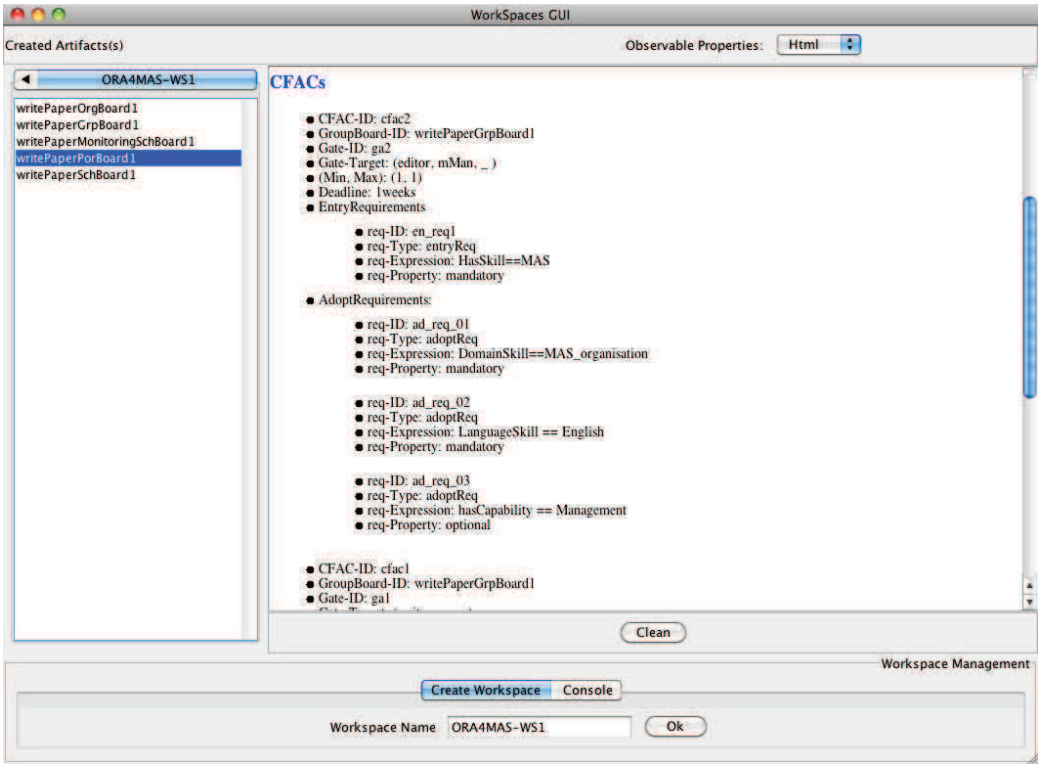
périmentations que nous avons fait.

Enfin, nous avons montré comment sont développés les agents qui nous ont servi dans nos expérimentations. Ils ont été développés avec la langage Jason. Nous avons présenté une simulation de création d'une entité organisationnelle dont la spécification organisationnelle est celle de l'exemple write-paper, que nous avons présenté dans les chapitres 5 et 6. Celle simulation illustre la création des artefacts organisationnels des des appels à candidatures, les procédures d'entrée, les engagements aux missions, une stratégie de régulation d'une transgression de norme et la coopération à la la réalisation des buts d'un schéma social.

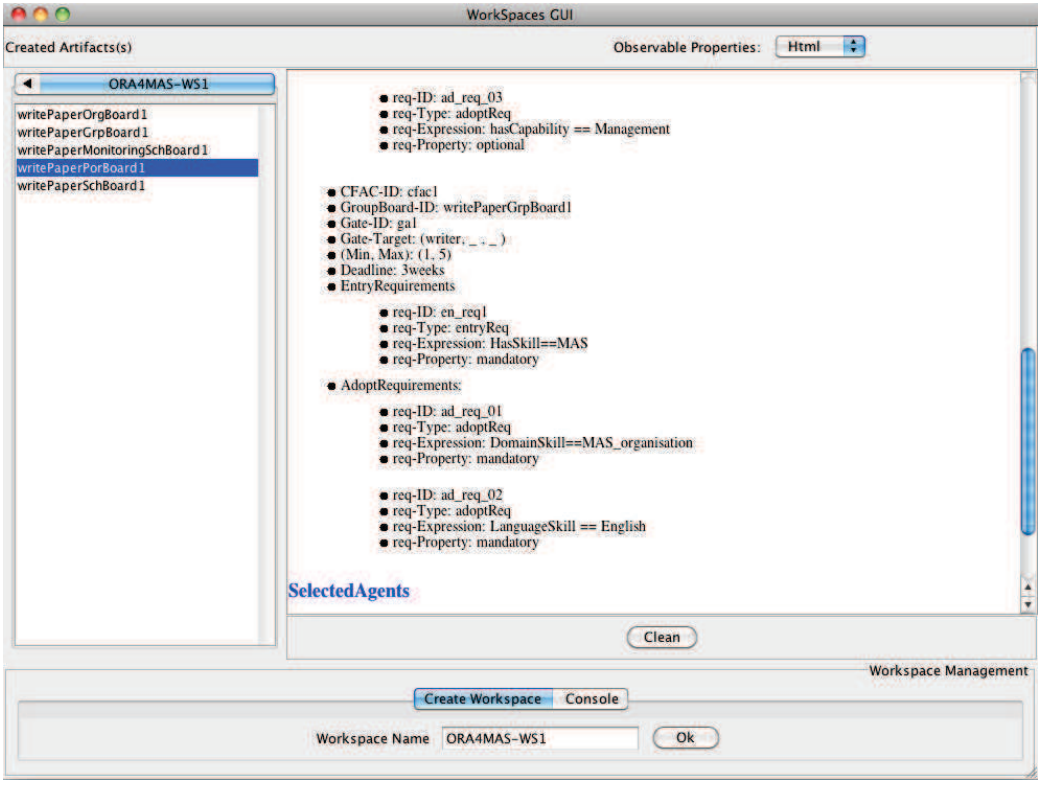
Nous pouvons donc résumer les intérêts d'utiliser les artefacts organisationnels pour la gestion d'organisations multi-agents ouvertes ci-dessous : (1) Permettre l'hétérogénéité des agents (par héritage des avantages de l'API CArtAgo) qui peuvent participer au cycle de vie (création, gestion des processus d'entrée / sortie, réalisation des buts des schémas sociaux, régulation) d'une entité organisationnelle. (2) Rendre accessible les spécifications en particulier celles des processus d'entrée sortie au moyen des propriétés observables. (3) Réduire le code des agents (en réduisant les envois de messages entre agents) pour leur coordination et pour la régulation avec le mécanisme de perception des évènements générés par les artefacts.

Primitives	**Définitions**
joinWorkspace(WsName, WsLocation)	Permet à un agent d'entrer dans le workspace WsName situé à l'adresse WsLocation
quitWorkspace(WsName)	Permet de sortir du workspace WsName
makeArtifact(ArName, ArType, ArID)	Permet de créer un artefact qui aura le nom ArName et le type ArType et enregistre son identifiant dans la variable ArID
lookupArtifactID(ArName, ArID)	Permet de chercher l'artefact dont le nom est ArName dans le workspace et enregistre son identifiant dans la variable ArID
disposeArtifact(ArID)	Permet de supprimer l'artefact dont l'identifiant est ArID
use(ArID, OpName(Params), SenID)	Permet d'exécuter sur l'artefact dont l'identifiant est ArID l'opération OpName avec les paramètre(s) Params et d'enregistrer les évènements générés dans la variable SenID
observeProperty(PropName, ArID)	Permet d'obtenir les données de la propriété observable PropName de l'artefact ArID
sense(SenID, Filter, Timeout)	Permet de capter un évènement selon le filtre Filter à partir des évènement enregistrés dans SenID après un délai Timeout
focus(ArID)	Permet de capter tous les évènements générés par l'artefact ArID
stopFocus(ArID)	Permet de capter tous les évènements générés par l'artefact ArID

TABLE 8.1 – Primitives utilisées par les agents pour manipuler les artefacts

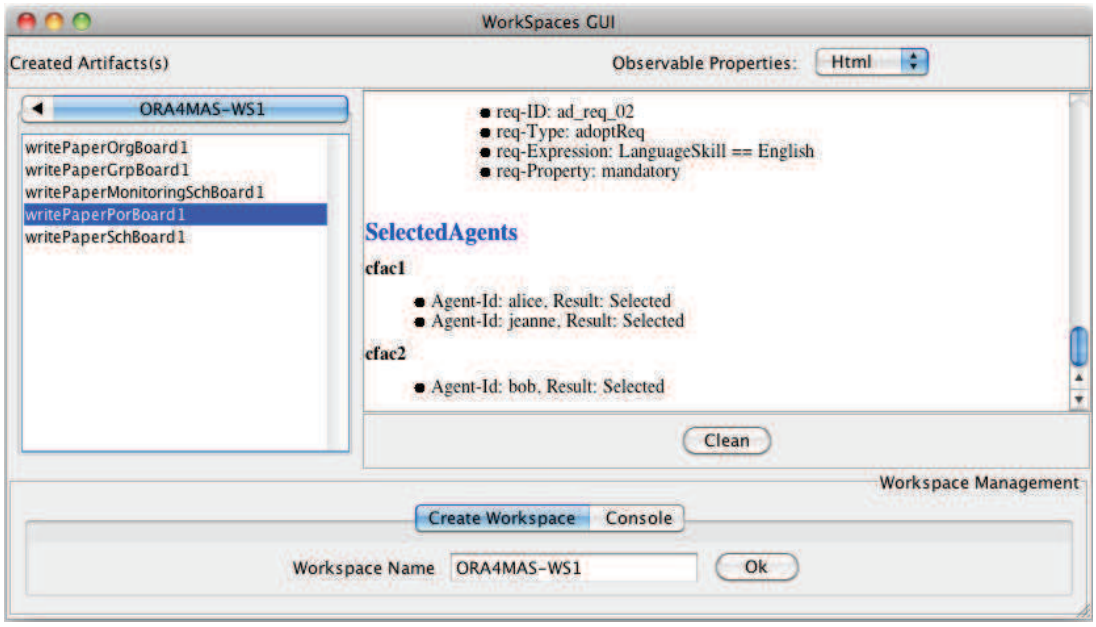


(a) Propriétés observables "CFACs" (1)

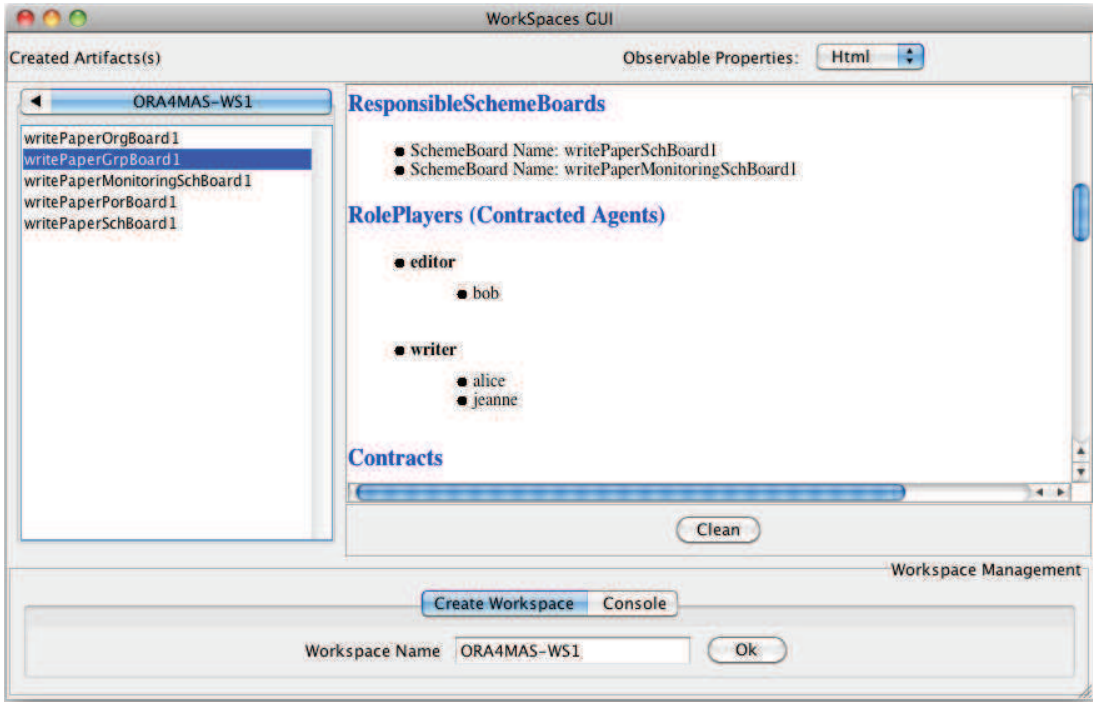


(b) Propriétés observables "CFACs" (2)

FIGURE 8.6 – Propriétés observables "CFACs" du PoratalBoard pour l'exemple "write-paper"

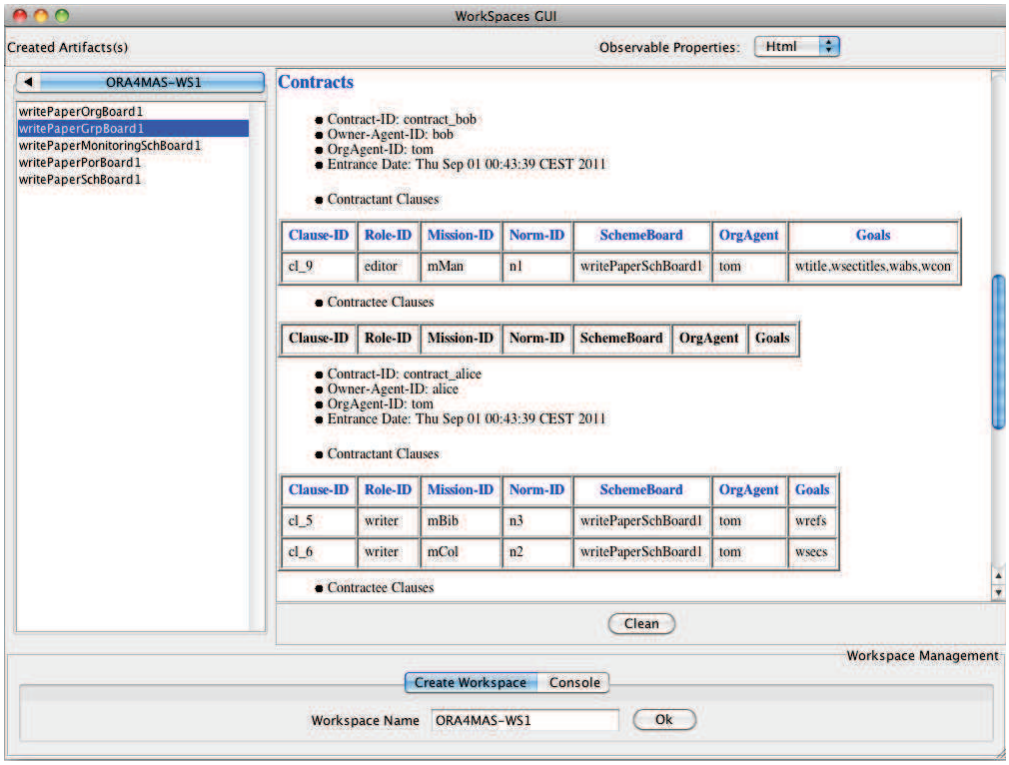


(a) Propriétés observables "SelectedAgents"

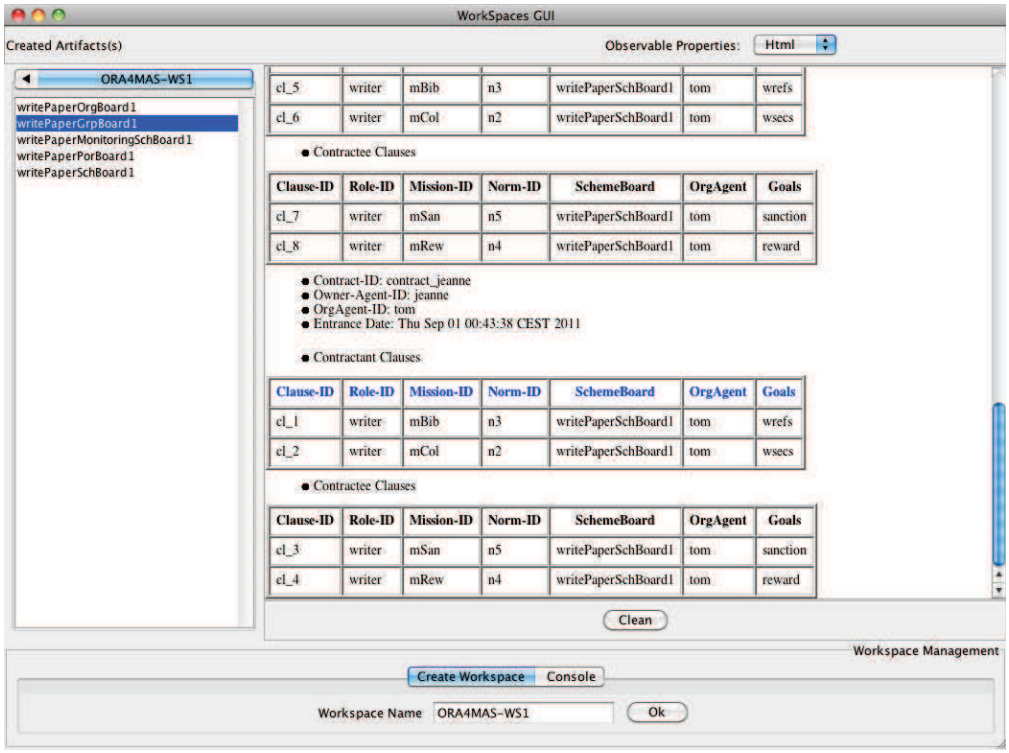


(b) Propriétés observables "RolePlayers"

FIGURE 8.7 – Propriétés observables du "SelectedAgents" PoratalBoard et "RolePlayers" GroupBoard pour l'exemple "write-paper"

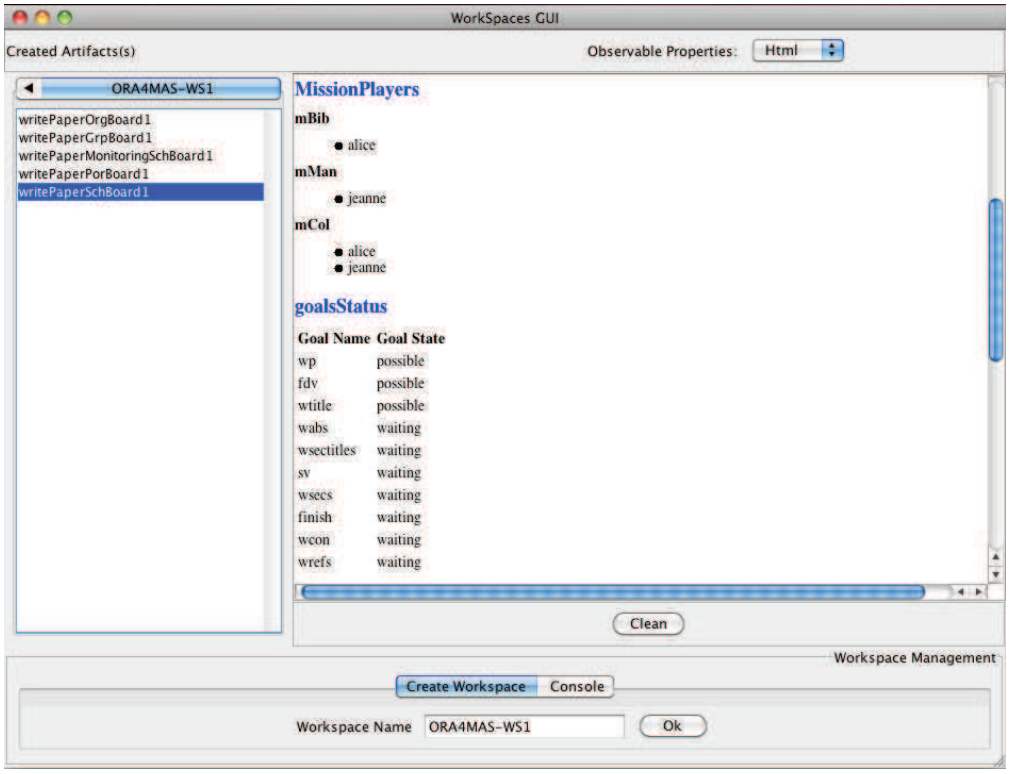


(a) Propriétés observables "Contracts" pour les agents bob et alice

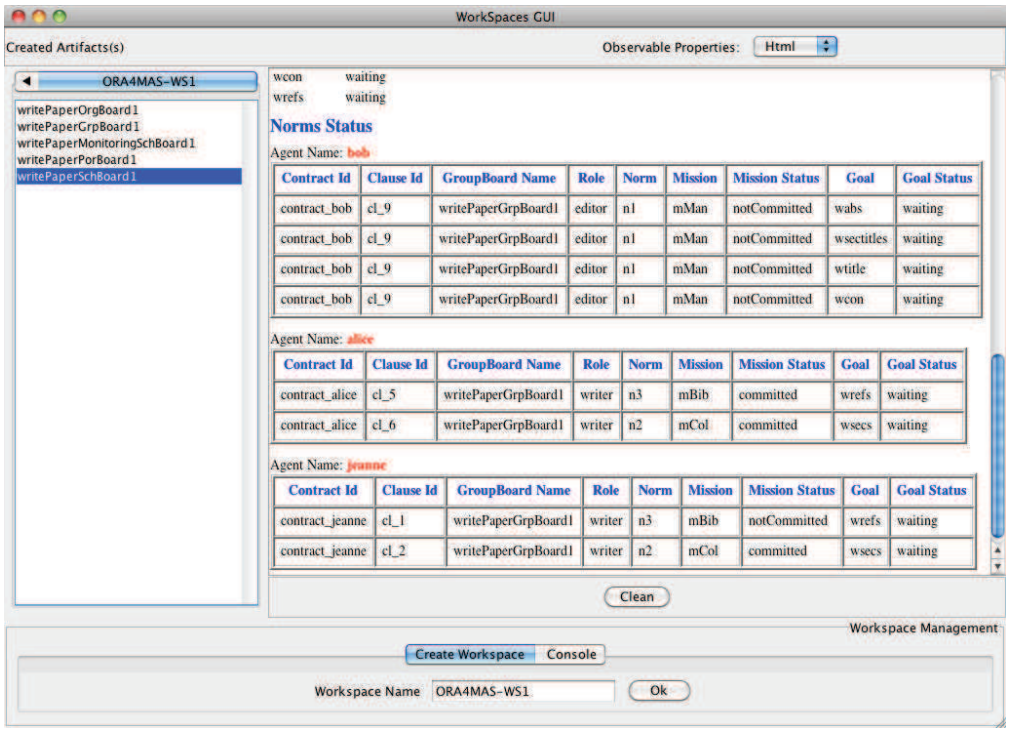


(b) Propriétés observables "Contracts" pour les agents alice et jeanne

FIGURE 8.8 – Propriétés observables du GroupBoard pour l'exemple "write-paper"

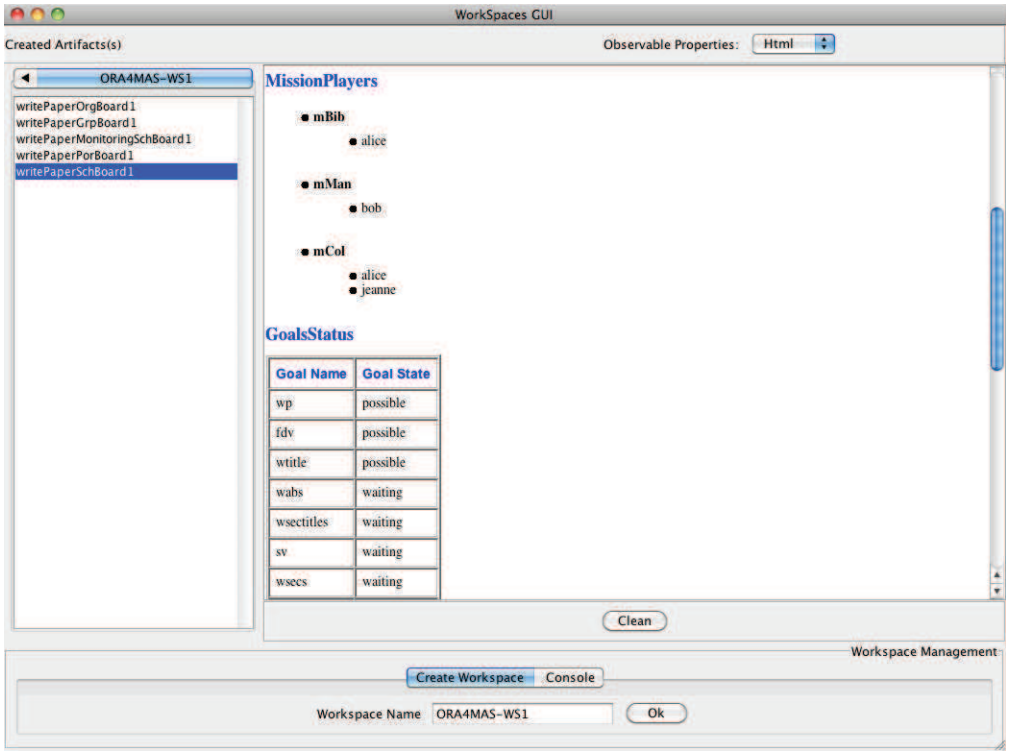


(a) Propriétés observables "MissionPlayers" pour les agents bob, alice et jeanne

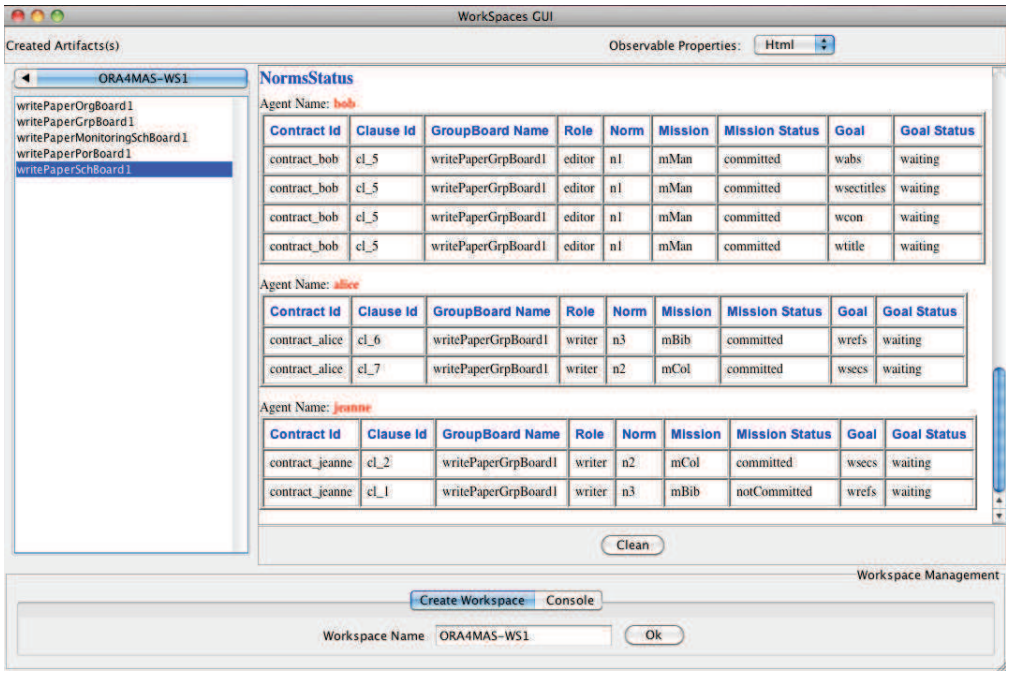


(b) Propriétés observables "NormStatus" pour les agents bob, alice et jeanne

FIGURE 8.9 – Propriétés observables du SchemeBoard pour l'exemple "write-paper"

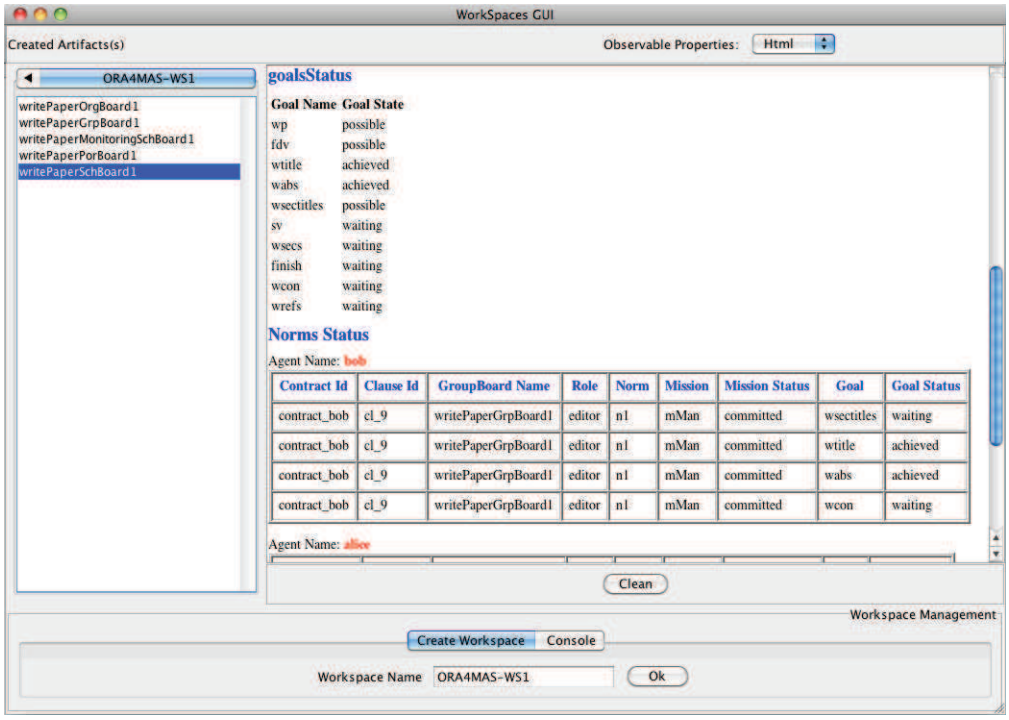


(a) Propriétés observables "MissionPlayers" pour bob, alice et jeanne après régulation

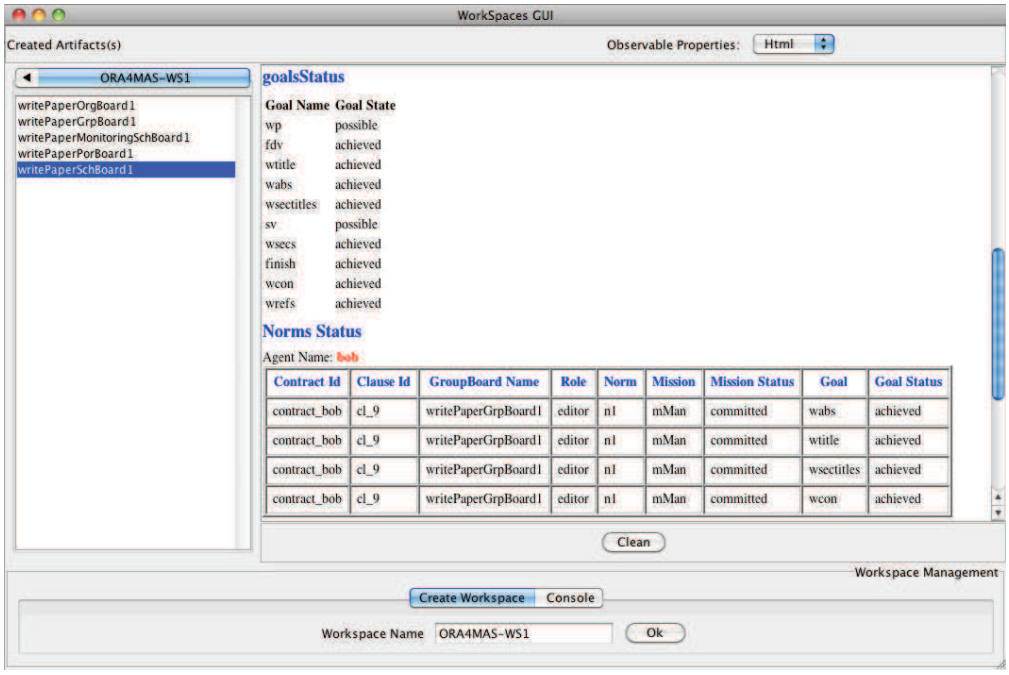


(b) Propriétés observables "NormStatus" pour bob, alice et jeanne après régulation

FIGURE 8.10 – Propriétés observables du SchemeBoard pour l'exemple "write-paper"



(a) Propriétés observables GoalStatus, NormStatus (1)



(b) Propriétés observables GoalStatus, NormStatus (2)

FIGURE 8.11 – Propriétés observables du SchemeBoard pour l'exemple "write-paper"

Chapitre 9

Cas d'étude et expérimentations

Dans ce chapitre nous allons illustrer nos propositions présentées dans les chapitres 5, 6 et 7 avec une organisation qui a pour objectif la construction d'un édifice. Cet exemple est une extension de l'exemple développé dans le cadre de EASSS 2010 du cours MAOP [1]. Dans un premier temps nous allons décrire en quoi consiste de façon simplifiée la construction d'un édifice. Nous présentons ensuite la spécification organisationnelle que nous avons conçue pour cette application. Nous terminerons le chapitre en présentant comment sont gérés concrètement les mécanismes de mise en oeuvre des propriétés d'ouverture (interopérabilité, entrée / sortie et contrôle) que nous proposons dans cette thèse.

9.1 Présentation générale de l'application "construction d'un édifice"

Notre objectif n'étant pas de faire une présentation exhaustive de tous les détails relatifs à une construction d'édifice, nous nous contenterons de quelques étapes relatives à une telle entreprise pour illustrer les représentations structurelle, fonctionnelle, normative et d'entrée/sortie d'une telle application selon le modèle MOISE que nous avons défini afin d'exprimer la coordination entre les différents acteurs/corps de métier tels que maçons, charpentiers, plombiers, électriciens, ...

Ainsi, nous supposons que pour l'édifice à construire, nous disposons du terrain pour la construction. Les principaux travaux pour la construction d'un édifice sont : le terrassement du terrain, la construction des fondations, l'élévation des murs, la

1. http ://easss2010.emse.fr/

construction de la charpente et du toit, les installations électriques, plomberies et des portes et fenêtres, le revêtement des murs intérieurs, des façade, du plafond et du sol.

La réalisation de tous ces travaux est décomposée en trois grandes étapes qui sont le gros oeuvre, le second oeuvre ou aménagement et les finitions.

Étape 1 : le gros oeuvre

Cette étape regroupe les travaux de préparation du terrain à construire et de la construction de la structure de l'édifice. On y retrouve le terrassement, la construction des fondations et éventuellement du sous sol, et éventuellement la construction de la structure : murs et/ou piliers porteurs de l'édifice.

Étape 2 : le second oeuvre

Le second oeuvre rassemble tous les travaux qui visent à aménager les différentes pièces d'un édifice de façon à les préparer pour les travaux de finitions. Il s'agit de construire les cloisons de l'édifice, d'installer l'électricité, de faire les travaux de plomberie, d'installer les portes et les fenêtres, de faire les travaux de préparation des revêtements du sol, du plafond et des murs.

Étape 3 : les finitions

Les travaux de finitions sont destinés à rendre un édifice habitable. Ils concernent le revêtement du plafond, le revêtement du sol, le revêtement des murs intérieurs et des façades et les finitions des travaux d'électricité et de plomberie.

9.2 Spécification organisationnelle de l'application avec MOISE

Après avoir décrit dans la section précédente notre cas d'étude, nous présentons ci-dessous sa spécification avec le langage de modélisation MOISE présenté dans le chapitre 5.

Afin d'illustrer notre approche de gestion de l'ouverture, notamment les procédures d'entrées / sorties que nous avons définies dans les chapitres 5 et 6, nous avons retenu quatre catégories d'exigences sur lesquelles se basera l'analyse des demandes d'entrée dans l'organisation.

Ainsi, nous avons des exigences qui concernent les *ressources (agents -humaines- et matérielles)* que chaque candidat à l'entrée dans l'organisation se propose de déployer pour la réalisation des buts correspondants à la porte d'entrée pour laquelle il

candidate. Nous avons associé cette catégorie d'exigences aux différents rôles qui ont été définis dans la SS de notre cas d'étude.

La deuxième catégorie d'exigences concerne la *démarche qualité* des candidats à l'entrée. Cette catégorie d'exigences regroupe pour certains buts et missions à réaliser dans l'organisation et définis dans la FS, quelques standards ou plus exactement, les normes française définies pour les tâches relatives aux buts ou missions concernés.

Les exigences définissant le coût de chaque but de la FS constituent la troisième catégorie d'exigence. Cette dernière permet à chaque candidat à l'entrée de préciser son évaluation du coût de chacun des buts associés à la porte d'entrée à laquelle il dépose sa candidature..

La dernière catégorie d'exigences regroupent celles qui constituent les exigences d'entrée dans l'organisation, les trois catégories précédemment citées étant des exigences d'adoption. Il s'agit de la taille de l'entreprise candidate à l'entrée ainsi que de ses références. Appartiennent également à cette catégorie, la démarche sécurité adoptée par l'entreprise candidate pour ses employés travaillant sur un chantier ainsi que la démarche environnementale utilisée par l'entreprise candidate lors de ses interventions sur un chantier.

9.2.1 Spécification structurelle

La spécification structurelle (SS) de notre cas d'étude est composée de quatre groupes, dont le groupe racine et le groupe $building\_grp$. Les autres groupes $bigworks\_grp$, $fitting\_grp$ et $finition\_grp$ sont des sous-groupes de $building\_grp$ (cf. Figure 9.1. Parmi les rôles on a ($r_1 : house\_owner$) qui est celui du propriétaire de l'édifice à construire, ($r_0 : building\_company$) qui est un rôle abstrait permettant de regrouper les caractéristiques générales des rôles qui représentent les métiers intervenant dans la construction d'un édifice. Par exemple, le rôle $r_2 : site\_prep\_contractor$ est le rôle représentant les travaux de terrassement. Un maçon est représenté par le rôle $r_3 : bricklayer$ et est chargé de la construction des fondations des mûrs porteurs et des poutres de l'édifice. Le couvreur ($r_4 : roofer$) s'occupe de la charpente et de la toiture. L'électricien et le plombier ($r_7 : electrician, r_6 : plumber$) comme leurs noms respectifs l'indiquent sont chargés des travaux d'électricité et de plomberie. Le plâtrier ($r_5 : platerer$) s'occupe de la construction des cloisons, des travaux d'isolation, du plafond et de la peinture. Le menuisier ($r_8 : joiner$) installe les ouvertures (portes et fenêtres) d'un édifice. Le carreleur ($r_9 : tiles\_layer$) est responsable de la pose des carreaux. Le façadier ($r_{10} : frontage\_maker$) est responsable de l'aménagement des façades de l'édifice.

La représentation graphique de la SS est présentée par la figure 9.1 et le résumé de la spécification des rôles est donné par le tableau 9.1.

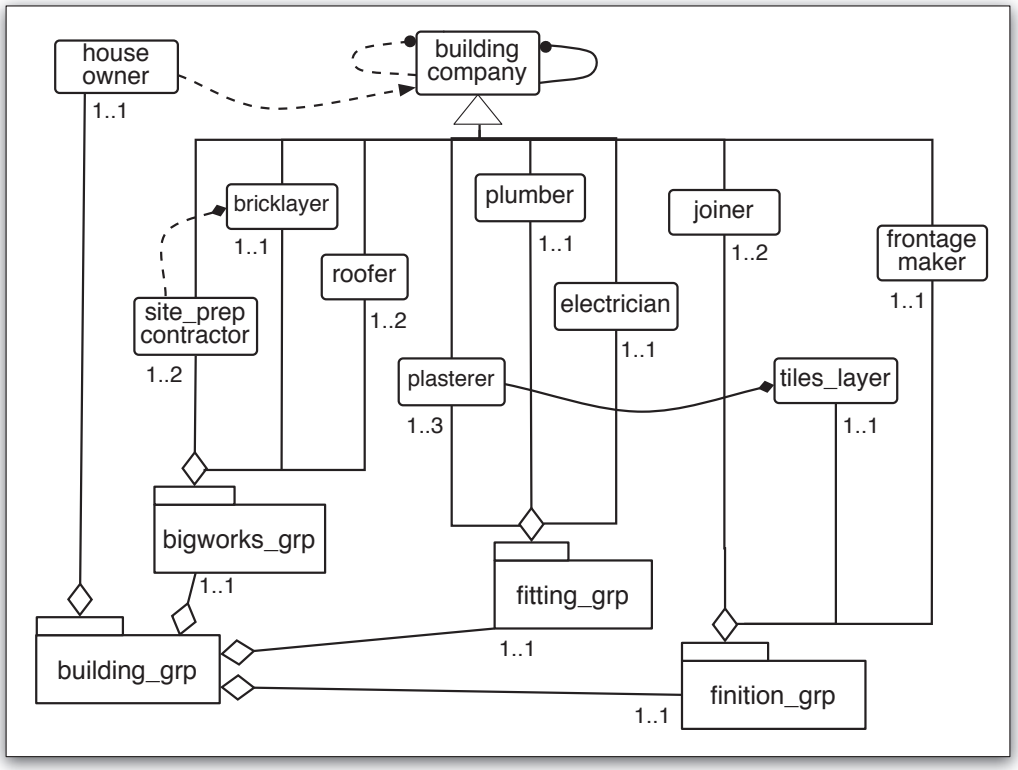


FIGURE 9.1 – Spécification structurelle de l'application « building house »

Selon la description formelle d'une SS présentée dans le chapitre 5, la SS de notre cas d'étude est spécifiée comme suit :

$$SS = \langle R, Gr, \sqsubset \rangle$$

- $Gr = \{building\_grp, bigworks\_grp, fitting\_grp, finition\_grp\}$: l'ensemble des groupes de la SS.
- $R = \{r_0, r_1, r_2, r_3, r_4, r_5, r_7, r_8, r_9, r_{10}\}$: l'ensemble des rôles définis pour la SS du cas d'étude. La tableau 9.1 présente la correspondance entre les indices $r_i \in R, (0 < i < 10)$ des rôles et les noms des rôles présentés dans la figure 9.1.

– $\sqsubset$: la relation d'héritage entre certains rôles de la SS définie comme suit : $\forall r \in R \backslash (R_{abs} \cup r_1), r \sqsubset r_0$, $R_{abs} = \{r_0\}$ est l'ensemble des rôles abstraits de la SS.

Spécification du groupe $building\_grp$

Avant de présenter la spécification du groupe $building\_grp$, rappelons la représentation formelle d'un groupe selon le langage MOISE présentée dans le chapitre 5.

$$\forall gr \in Gr, gr = \langle R_{gr}, SG_{gr}, L_{gr}^{intra}, L_{gr}^{inter}, \mathcal{C}_{gr}^{intra}, \mathcal{C}_{gr}^{inter}, ng_{gr}, nr_{gr} \rangle$$

Un groupe est donc composé de l'ensemble de ses rôles, de l'ensemble de ses sous-groupes, des ensembles de ses liens intra-groupe et inter-groupe, des ensembles de ses contraintes de compatibilités intra-groupe et inter-groupe, des fonctions permettant respectivement de définir les cardinalités de ses sous-groupes et celles de ses rôles. Ainsi, la spécification du groupe $building\_grp$ est la suivante :

$$building\_grp = \langle R_{building\_grp}, SG_{building\_grp}, L_{building\_grp}^{intra}, \phi, \phi, \phi, ng_{building\_grp}, nr_{building\_grp} \rangle$$

– $R_{building\_grp} = \{r_1\}$

– $SG_{building\_grp} = \{bigworks\_grp, fitting\_grp, finition\_grp\}$

– $L_{building\_grp}^{intra} = \{l_1, l_2\}$
  – $l_1 = \langle r_1, r_0, aut \rangle$
  – $l_2 = \langle r_0, r_0, com \rangle$

– La cardinalité minimale et maximale de chacun des sous-groupes de $building\_grp$ définie par la fonction $ng_{building\_grp}$ est donnée ci-dessous :

$$ng_{building\_grp}(bigworks\_grp) = (1, 1)$$
$$ng_{building\_grp}(fitting\_grp) = (1, 1)$$
$$ng_{building\_grp}(finition\_grp) = (1, 1)$$

– La fonction $nr_{building\_grp}$ fourni la cardinalité minimale et maximale du seul rôle r_1 du groupe $building\_grp$: $nr_{building\_grp}(r_1) = (1, 1)$

Comme nous l'avons dit ci-dessus, $building\_grp$ est le groupe racine de notre cas d'étude. Il comprend l'unique rôle r_1 et tous les autres groupes de la SS sont ses sous-groupes. L'ensemble de ses liens inter-groupe est vide et il a deux liens intra-groupe. Le premier est un lien d'autorité entre le rôle r_1 et le rôle abstrait r_0, et le second est

un lien de communication entre r_0 et r_0. Puisque r_0 est un rôle abstrait alors tous les rôles qui héritent de r_0 héritent de ce lien de communication. De même, le rôle r_1 a autorité sur tout rôle qui hérite de r_0. Les ensembles des contraintes de compatibilité intra-groupe et inter-groupe de $building\_grp$ sont vides.

Spécification du groupe $bigworks\_grp$

Le groupe $bigworks\_grp$ est formellement défini ci-dessous. Il comprend trois rôles r_2, r_3, r_4. Contrairement au groupe $building\_grp$, il n'a pas de sous-groupe et ses ensembles de liens inter-groupe et intra-groupe sont vides ainsi que l'ensemble de contraintes de compatibilité inter-groupe. Cependant, ses rôles héritent des liens de communication intra-groupe et inter-groupe définis sur le rôle abstrait r_0. Il existe une contrainte de compatibilité intra-groupe qui est définie entre les rôles r_2 et r_3. Ainsi, tout agent jouant le rôle $r_2 : site\_prep\_contractor$ peut également jouer le rôle $r_3 : bricklayer$ dans l'instance de groupe des gros travaux ($bigworks\_grp$) de construction d'un édifice. La cardinalité respective de chaque rôle est donnée par la fonction $nr_{bigworks\_grp}$.

$bigworks\_grp = \langle R_{bigworks\_grp}, \phi, \phi, \mathcal{C}^{intra}_{bigworks\_grp}, \phi, ng_{bigworks\_grp}, nr_{bigworks\_grp} \rangle$

- $R_{bigworks\_grp} = \{r_2, r_3, r_4\}$
- $\mathcal{C}^{intra}_{bigworks\_grp} = \{comp_1\}$
  - $comp_1 = r_2 \bowtie r_3$
- Valeur de $nr_{bigworks\_grp}$ pour chaque rôle.
  $nr_{bigworks\_grp}(r_2) = (1, 2)$
  $nr_{bigworks\_grp}(r_3) = (1, 1)$
  $nr_{bigworks\_grp}(r_4) = (1, 2)$
- Puisque le groupe $bigworks\_grp$ n'a pas de sous-groupe, la fonction $ng_{bigworks\_grp}$ n'est définie pour aucune valeur.

Spécification du groupe $fitting\_grp$

De même que le groupe $bigworks\_grp$, $fitting\_grp$ n'a pas de sous-groupe et ses ensembles de liens inter-groupe et intra-groupe sont vides. Ses trois rôles qui héritent des liens de communication intra-groupe et inter-groupe définis sur le rôle abstrait r_0. Il existe une contrainte de compatibilité inter-groupe entre son rôle r_5 et le rôle r_9 du groupe $finition\_grp$. L'ensemble de ses contraintes de compatibilité intra-groupe est vide et puisqu'il n'a pas de sous-groupe, la fonction $ng_{fitting\_grp}$ n'est définie pour

aucune valeur. Sa spécification selon le langage MOISE est la suivante :

$$fitting\_grp = \langle R_{fitting\_grp}, \phi, \phi, \phi, \phi, ng_{fitting\_grp}, nr_{fitting\_grp} \rangle$$

- $R_{fitting\_grp} = \{r_5, r_6, r_7\}$
- $\mathcal{C}^{inter}_{fitting\_grp} = \{comp_2\}$
  - $comp_2 = r_5 \bowtie r_9$
- Valeur de $nr_{fitting\_grp}$ pour chaque rôle.
  $nr_{fitting\_grp}(r_5) = (1, 3)$
  $nr_{fitting\_grp}(r_6) = (1, 1)$
  $nr_{fitting\_grp}(r_7) = (1, 1)$

Spécification du groupe $finition\_grp$

Le groupe $finition\_grp$ a trois rôles r_8, r_9, r_{10} dont la cardinalité de chacun est donnée par la fonction $nr_{finition\_grp}$. Comme tous les sous-groupes du groupe racine $builiding\_grp$ de la SS de notre cas d'étude, il n'a pas de sous-groupe. En conséquence, la fonction $ng_{finition\_grp}$ n'est définie pour aucune valeur. $finition\_grp$ n'a pas de lien intra-groupe, ni inter-groupe. Mais comme pour les groupes $bigworks\_grp$ et $fitting\_grp$ présentés ci-dessus, ses rôles héritent des liens de communication intra-groupe et inter-groupe définis sur le rôle abstrait r_0. Il n'a également pas de contrainte de compatibilité intra-groupe et inter-groupe.

$$finition\_grp = \langle R_{finition\_grp}, \phi, \phi, \phi, \phi, ng_{finition\_grp}, nr_{finition\_grp} \rangle$$

- $R_{finition\_grp} = \{r_8, r_9, r_{10}\}$
- Valeur de $nr_{finition\_grp}$ pour chaque rôle.
  $nr_{finition\_grp}(r_8) = (1, 2)$
  $nr_{finition\_grp}(r_9) = (1, 1)$
  $nr_{finition\_grp}(r_{10}) = (1, 1)$

Spécification des exigences des rôles

Les rôles d'une SS peuvent être attachés à des concepts qui peuvent être des portes d'entrée dans une organisation. Ainsi, à certains rôles de notre cas d'étude sont associés des exigences d'engagement et des exigences d'abandon. En particulier, pour chaque rôle qui hérite de r_0 : $building\_company$ deux exigences d'engagements sont définies : le nombre de ressources humaines et la liste de ressources matérielles qui

Role-Id	**Intitulé**	$cardinality$	AdoptReq	LeaveReq
r_0	building_company			
r_1	house_owner	(1, 1)	ϕ	ϕ
r_2	site_prep_contractor	(1, 2)	ar_1r_2, ar_2r_2	ϕ
r_3	bricklayer	(1, 1)	ar_1r_3, ar_2r_3	ϕ
r_4	roofer	(1, 2)	ar_1r_4, ar_2r_4	ϕ
r_5	plasterer	(1, 3)	ar_1r_5, ar_2r_5	ϕ
r_6	plumber	(1, 1)	ar_1r_6, ar_2r_6	ϕ
r_7	electrician	(1, 1)	ar_1r_7, ar_2r_7	ϕ
r_8	joiner	(1, 2)	ar_1r_8, ar_2r_8	ϕ
r_9	tiles_layer	(1, 1)	ar_1r_9, ar_2r_9	ϕ
r_{10}	frontage_maker	(1, 1)	ar_1r_{10}, ar_2r_{10}	ϕ

TABLE 9.1 – Rôles de l'application « house_building »

seront déployées par le candidat pour la réalisation des engagements relatifs au rôle. Avant de présenter les exigences d'engagement et d'abandon des rôles de notre cas d'étude, nous rappelons ci-dessous la représentation d'une exigence présenté dans la section 5.2.

$$
\begin{aligned}
Requirement &::= \langle type, RExpr, property\rangle \\
type &::= EntryReq|AdoptReq|LeaveReq|ExitReq \\
RExpr &::= SRExpr \mid (RExpr\ '\wedge'\ RExpr) \mid (RExpr\ '\vee'\ RExpr) \\
SRExpr &::= (Term\ op\ Value) \mid (Term\ \ '==\ '\ \ '\_') \\
op &::= \ '==\ ' \ \mid \ '\leqslant' \ \mid \ '\geqslant' \ \mid \ '<' \ \mid \ '>' \ \mid \ '\neq' \\
property &::= mandatory|optional
\end{aligned}
$$

Une exigence qui concerne le nombre de ressources humaines (team_memb_numb) pour un rôle r_i est notée ar_1r_i, tandis qu'une exigence qui concerne la liste de ressources matérielles est notée ar_2r_i. La représentation générale de chacune de ces exigences est la suivante :

$$
\begin{aligned}
ar_1r_i = \langle AdoptReq, team\_memb\_numb \leq v_1 \quad \wedge \\
team\_memb\_numb \geq v_2, mand\rangle,\ v_1, v_2 \in \mathbb{N} \\
ar_2r_i = \langle AdoptReq, material\_provided == \_, mandatory\rangle
\end{aligned}
$$

La représentation de ar_1r_i permet de préciser un intervalle de valeur du nombre de personnel souhaité pour chaque rôle. Quant à la représentation de ar_2r_i puisqu'il revient à chaque agent candidat de donner la liste de matériel qu'il déploiera pour la réalisation des engagements du rôle considéré, l'expression de l'exigence se résume à $material\_provided == \_$ qui signifie que la valeur de $material\_provided$ n'est pas prédéfinie.

Pour tous les rôles $r_2, ..., r_{10}$ qui héritent de r_0 nous ne re-préciserons pas ici leur exigence ar_2r_i puisque c'est la même représentation quelque soit le rôle. Ainsi, $ar_2r_2 = ... = ar_2r_{10} = \langle AdoptReq, material\_provided == \_, mandatory \rangle$.

Liste des exigences ar_1r_i pour les rôles $r_2, ..., r_{10}$

$$ar_1r_2 = \langle AdoptReq, team\_numb \leq 5 \ \wedge \ team\_numb \geq 3, mandatory \rangle$$

$$ar_1r_3 = \langle AdoptReq, team\_numb \leq 7 \ \wedge \ team\_numb \geq 5, mandatory \rangle$$

$$ar_1r_4 = \langle AdoptReq, team\_numb \leq 10 \ \wedge \ team\_numb \geq 4, mandatory \rangle$$

$$ar_1r_5 = \langle AdoptReq, team\_numb \leq 5 \ \wedge \ team\_numb \geq 2, mandatory \rangle$$

$$ar_1r_6 = \langle AdoptReq, team\_numb \leq 3 \ \wedge \ team\_numb \geq 2, mandatory \rangle$$

$$ar_1r_7 = \langle AdoptReq, team\_numb \leq 6 \ \wedge \ team\_numb \geq 1, mandatory \rangle$$

$$ar_1r_8 = \langle AdoptReq, team\_numb \leq 5 \ \wedge \ team\_numb \geq 3, mandatory \rangle$$

$$ar_1r_9 = \langle AdoptReq, team\_numb \leq 7 \ \wedge \ team\_numb \geq 4, mandatory \rangle$$

$$ar_1r_{10} = \langle AdoptReq, team\_numb \leq 10 \ \wedge \ team\_numb \geq 4, mandatory \rangle$$

9.2.2 Spécification fonctionnelle

La spécification fonctionnelle (FS) d'une simulation de construction d'un édifice comprend les schémas sociaux qui décrivent la façon dont les agents vont se coordonner pour mener à bien les missions et les buts spécifiques aux travaux de constructions. Sa spécification selon le langage MOISE est définie comme suit.

$$FS = \langle S, Pr \rangle$$

Aucune préférence Pr n'est définie sur les missions du schéma social S de ce cas d'étude. La description détaillée de S selon le langage MOISE est présentée ci-dessous. Elle fournit les ensembles des buts (G_{bhs}), des missions (M_{bhs}), et des plans

(P_{bhs}) du schéma social, ainsi que les fonctions qui définissent les cardinalités des missions (nm_{bhs}), des buts (ng_{bhs}), ainsi que la fonction mo_{bhs} qui définit le sous ensemble de buts d'une mission. La définition correspondante aux identifiants des missions de la spécification ci-dessous est présentée dans le tableau 9.2 et celle correspondante aux identifiants de buts est fournie dans la tableau 9.3. La représentation graphique du schémas social est présentée par la figure 9.2.

$$
\begin{aligned}
&S = \{build\_house\_sch\}\\
&build\_house\_sch = \langle G_{bhs}, M_{bhs}, P_{bhs}, mo_{bhs}, nm_{bhs}, ng_{bhs}\rangle\\
&\quad G_{bhs} = G_{bhs}^{nodes} \cup G_{bhs}^{leaves}\\
&\qquad G_{bhs}^{nodes} = \{g_0, g_{01}, g_{02}, g_{03}, g_{04}, g_{11}, g_{12}, g_{13}, g_{14}, g_{15}\}\\
&\qquad G_{bhs}^{leaves} = \{g_{1a}, g_{1b}, g_{2a}, g_{2b}, g_{3a}, g_{3b}, g_{4a}, g_{4b}, g_{4c}, g_{4d},\\
&\qquad\qquad g_{5a}, g_{5b}, g_{5c}, g_{6a}, g_{6b}, g_{6c}, g_{7a}, g_{7b}, g_{8a}, g_{9a}\}\\
&\quad M_{bhs} = \{m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10}\}\\
&\quad P_{bhs} = \{p_0, p_1, p_{11}, p_{12}, p_{13}, p_{14}, p_2, p_{21}, p_{22}, p_2\}\\
&\qquad p_0 = \langle g_0, \mathbf{,}, G_{p_0}\rangle,\ G_{p_0} = \{g_{01}, g_{03}, g_{04}\}\\
&\qquad p_1 = \langle g_{01}, \mathbf{,}, G_{p_1}\rangle,\ G_{p_1} = \{g_{1a}, g_{11}, g_{12}, g_{02}\}\\
&\qquad p_{11} = \langle g_{11}, \|, G_{p_{11}}\rangle,\ G_{p_{11}} = \{g_{2a}, g_{1b}\}\\
&\qquad p_{12} = \langle g_{12}, \|, G_{p_{12}}\rangle,\ G_{p_{12}} = \{g_{2b}, g_{5a}, g_{6a}\}\\
&\qquad p_{13} = \langle g_{02}, \|, G_{p_{13}}\rangle,\ G_{p_{13}} = \{g_{13}, g_{4a}\}\\
&\qquad p_{14} = \langle g_{13}, \mathbf{,}, G_{p_{14}}\rangle,\ G_{p_{14}} = \{g_{3a}, g_{3b}\}\\
&\qquad p_2 = \langle g_{03}, \mathbf{,}, G_{p_2}\rangle,\ G_{p_2} = \{g_{14}, g_{15}\}\\
&\qquad p_{21} = \langle g_{14}, \|, G_{p_{21}}\rangle,\ G_{p_{21}} = \{g_{4b}, g_{5b}, g_{6b}\}\\
&\qquad p_{22} = \langle g_{15}, \|, G_{p_{22}}\rangle,\ G_{p_{22}} = \{g_{7a}, g_{7b}, , g_{9a}\}\\
&\qquad p_3 = \langle g_{04}, \mathbf{,}, G_{p_3}\rangle,\ G_{p_3} = \{g_{4c}, g_{8a}, g_{5c}, g_{6c}, g_{4d}\}
\end{aligned}
$$

Les définitions en extension des fonctions nm_{bhs} et mo_{bhs} sont présentées dans le tableau 9.2 ainsi que la liste des exigences d'engagement et d'abandon de chaque mission. La valeur de la fonction ng_{bhs} pour chaque but de G_{bhs} est présentée dans le tableau 9.3 ainsi que les exigences d'engagement et d'abandon de chaque but.

Spécification des exigences des missions

Une mission dans une organisation définie avec l'OML MOISE est composée

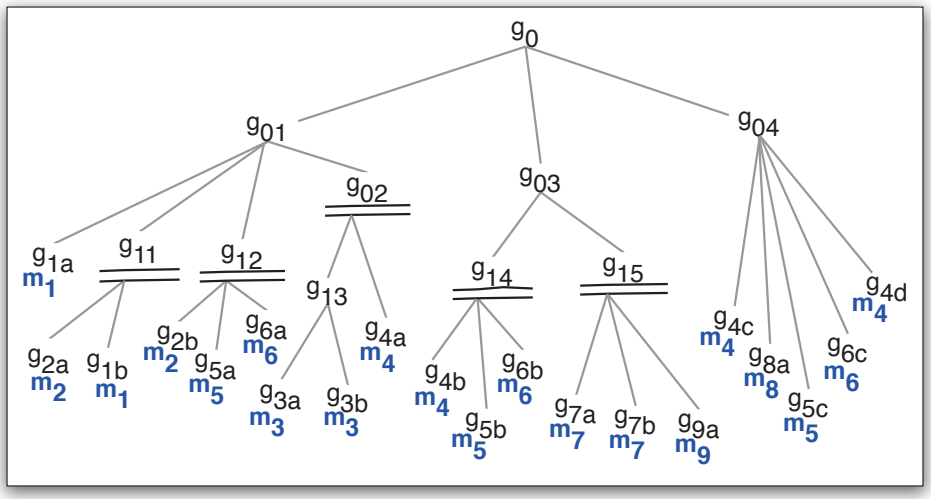


FIGURE 9.2 – Schéma social de l'application « building house »

d'un ensemble de buts que doit réaliser tout agent qui s'engage à la mission. Ainsi, les exigences d'engagements des missions et des buts de notre cas d'étude ont pour objectif de se prémunir de la bonne réalisation de l'objectif ou but racine du schémas social objectif qui n'est autre que la simulation de la construction d'un édifice. Cet objectif se décline en missions et buts de la spécification fonctionnelle de notre cas d'étude. Ainsi, les deux types d'exigences que nous avons définis pour notre cas d'étude sont (1) les standards français en vigueur pour les travaux de construction, (2) Le coût de réalisation des missions et buts.

Pour le cas particulier de spécification d'un standard comme exigence d'engagement pour une mission, il faut s'assurer que ce standard concerne tous les buts de la mission. Ainsi, plusieurs missions n'ont pas d'exigences de type standard car leurs buts étant très indépendants, il n'existe pas de standard pour toute la mission, mais des standards spécifiques à chaque but.

En ce qui concerne la spécification d'exigence de type coût de réalisation d'une mission, nous avons décidé qu'elle correspond à la somme du coût de réalisation de chaque but de la mission. Raison pour laquelle, nous n'avons pas défini ce type d'exigence pour les missions de notre cas d'étude. Cette exigence étant définie pour chaque but de la FS, et donc la valeur correspondante au coût de réalisation d'une mission pourra être calculée à partir des valeurs de réalisation de chacun de ses buts.

Nous présentons ci-dessous les exigences d'engagements qui ont été définies pour

Id	**Intitulé**	$nm_{build\_house\_sch}$	mo_{bhs}	AdoptReq	LeaveReq
m_0	manage_building	(1, 1)	g_0	ϕ	ϕ
m_1	prepare_site	(1, 2)	$\{g_{1a}, g_{1b}\}$	ϕ	ϕ
m_2	build_structure	(1, 1)	$\{g_{2a}, g_{2b}\}$	ϕ	ϕ
m_3	build_roof	(1, 2)	$\{g_{3a}, g_{3b}\}$	ϕ	ϕ
m_4	build_int-wall	(1, 3)	$\{g_{4a}, g_{4b}, g_{4c}, g_{4d}\}$	ϕ	ϕ
m_5	install_plumbing	(1, 1)	$\{g_{5a}, g_{5b}, g_{5c}\}$	$ar_1 m_5, ar_2 m_5$	ϕ
m_6	install_electricity	(1, 1)	$\{g_{6a}, g_{6b}, g_{6c}\}$	$ar_1 m_6$	ϕ
m_7	fit_door-windows	(1, 2)	$\{g_{7a}, g_{7b}\}$	ϕ	ϕ
m_8	fit_tiler	(1, 1)	$\{g_{8a}\}$	$ar_1 m_8$	ϕ
m_9	fit_frontage	(1, 1)	$\{g_{9a}\}$	ϕ	ϕ

TABLE 9.2 – Missions de l'application « house_building »

les missions m_5, m_6 et m_8. Les autres missions n'ont pas d'exigence d'engagement.

$$ar_1 m_5 = \langle AdoptReq, has\_standard == \text{DTU 60.11}, mandatory \rangle$$
$$ar_2 m_5 = \langle AdoptReq, has\_standard == \text{DTU 65}, mandatory \rangle$$
$$ar_1 m_6 = \langle AdoptReq, has\_standard == \text{NF C 15-100}, mandatory \rangle$$
$$ar_1 m_8 = \langle AdoptReq, has\_standard == \text{DTU 52.1}, mandatory \rangle$$

Spécification des exigences des buts

Toute exigence d'engagement à un but $g_{i\alpha}$ (cf. tableau 9.3) de type coût de réalisation du but est notée $ar_1 g_{i\alpha}$ et est représentée de la façon suivante.

$$ar_1 g_{i\alpha} = \langle AdoptReq, goal\_cost == \_, mandatory \rangle$$

Cette représentation exprime le fait que la valeur du coût d'un but représenté par le terme $goal\_cost$ n'est pas prédéfinie mais doit être fournie par tout agent qui candidate à l'engagement à ce but. Ainsi, pour chaque but $g_{i\alpha}$ du tableau 9.2) pour lequel une exigence $ar_1 g_{i\alpha}$, on considérera qu'elle s'exprime comme ci dessus.

Exemple, les exigences $ar_1 g_{1a}, ar_1 g_{1b}, ar_1 g_{4d}, ar_1 g_{5c}, ar_1 g_{9a}$ des buts $g_{1a}, g_{1b}, g_{4d}, g_{5c}, g_{9a}$ sont spécifiées comme suit :

$$ar_1g_{1a} = \langle AdoptReq, goal\_cost == \_, mandatory \rangle$$
$$ar_1g_{1b} = \langle AdoptReq, goal\_cost == \_, mandatory \rangle$$
$$ar_1g_{5c} = \langle AdoptReq, goal\_cost == \_, mandatory \rangle$$
$$ar_1g_{4d} = \langle AdoptReq, goal\_cost == \_, mandatory \rangle$$
$$ar_1g_{9a} = \langle AdoptReq, goal\_cost == \_, mandatory \rangle$$

Les exigences de type standard définies pour quelques buts de la FS sont présentées ci-dessous.

$$ar_2g_{3a} = \langle AdoptReq, has\_standard == \text{NF P 21-203-1}, mandatory \rangle$$
$$ar_2g_{3b} = \langle AdoptReq, has\_standard == \text{NF P 31-301}, mandatory \rangle$$
$$ar_2g_{7b} = \langle AdoptReq, has\_standard == \text{DTU 36.1}, mandatory \rangle$$
$$ar_2g_{9a} = \langle AdoptReq, has\_standard == \text{DTU 59.1}, mandatory \rangle$$

L'exigence d'abandon de chaque but notée lr_{gID} avec gID est l'identifiant du but. Elle est de la forme suivante :

$$lr_{gID} = \langle LeaveReq, hasFinishedGoal == \_, mandatory \rangle$$

Quelques exemples d'exigence d'abandon pour quelques buts de la FS sont ci-dessous présentée.

$$lr_{g1a} = \langle LeaveReq, hasFinishedGoal == \_, mandatory \rangle$$
$$lr_{g1b} = \langle LeaveReq, hasFinishedGoal == \_, mandatory \rangle$$
$$lr_{g5c} = \langle LeaveReq, hasFinishedGoal == \_, mandatory \rangle$$
$$lr_{g4d} = \langle LeaveReq, hasFinishedGoal == \_, mandatory \rangle$$
$$lr_{g9a} = \langle LeaveReq, hasFinishedGoal == \_, mandatory \rangle$$

Id	**intitulé**	$ng_{build\_house\_sch}$	ttf	AdoptReq	LeaveReq
g_{1a}	escavation	(1, 1)	1 week	$ar_1 g_{1a}$	lr_{g1a}
g_{1b}	roal_sewer_system	(1, 1)	3 week	$ar_1 g_{1b}$	lr_{g1b}
g_{2a}	fondation_built	(1, 1)	4 weeks	$ar_1 g_{2a}$	lr_{g2a}
g_{2b}	walls_built	(1, 1)	6 weeks	$ar_1 g_{2b}$	lr_{g2b}
g_{3a}	roof-structure_built	(1, 1)	2 weeks	$ar_1 g_{3a}, ar_2 g_{3a}$	lr_{g3a}
g_{3b}	roof_built	(1, 1)	2 weeks	$ar_1 g_{3b}, ar_2 g_{3b}$	lr_{g3b}
g_{4a}	steelframe_built	(1, 1)	1 week	$ar_1 g_{4a}$	lr_{g4a}
g_{4b}	isolation_made	(1, 1)	1 week	$ar_1 g_{4b}$	lr_{g4b}
g_{4c}	ceilling_made	(1, 1)	1 week	$ar_1 g_{4c}$	lr_{g4c}
g_{4d}	interior_painted	(1, 1)	1 week	$ar_1 g_{4d}$	lr_{g4d}
g_{5a}	pl_incorporation	(1, 1)	2 weeks	$ar_1 g_{5a}$	lr_{g5a}
g_{5b}	pl_installed	(1, 1)	4 weeks	$ar_1 g_{5b}$	lr_{g5b}
g_{5c}	pl_finitions	(1, 1)	1 weeks	$ar_1 g_{5c}$	lr_{g5c}
g_{6a}	el_incorporation	(1, 1)	2 weeks	$ar_1 g_{6a}$	lr_{g6a}
g_{6b}	el_installed	(1, 1)	4 weeks	$ar_1 g_{6b}$	lr_{g6b}
g_{6c}	el_finitions	(1, 1)	1 weeks	$ar_1 g_{6c}$	lr_{g6c}
g_{7a}	doors_fitted	(1, 1)	1 week	$ar_1 g_{7a}$	lr_{g7a}
g_{7b}	windows_fitted	(1, 1)	1 week	$ar_1 g_{7b}, ar_2 g_{7b}$	lr_{g7b}
g_{8a}	tiles_layed	(1, 1)	2 week	$ar_1 g_{8a}$	lr_{g8a}
g_{9a}	frontage_fitted	(1, 1)	2 week	$ar_{g9a}, ar_2 g_{9a}$	lr_{g9a}

TABLE 9.3 – Buts feuilles de l'application « house_building »

9.2.3 Spécification normative

Les normes qui régissent les droits et devoirs des rôles de la SS vis à vis des missions de la FS sont présentés dans le tableau 9.4.

- $NS = \langle N \rangle$
  - $\forall n \in N, n = \langle condition, role, type, mission, ttf \rangle$
- $N = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9, n_{10}\}$

n_{id}	condition	role	type	mission	TTF
n_0		r_1	$obligation$	m_0	–
n_1		r_2	$obligation$	m_1	3 weeks
n_2		r_3	$obligation$	m_2	10 weeks
n_3		r_4	$obligation$	m_3	4 weeks
n_4		r_5	$obligation$	m_4	6 weeks
n_5		r_6	$obligation$	m_5	3 weeks
n_6		r_7	$obligation$	m_6	3 weeks
n_7		r_8	$obligation$	m_7	3 weeks
n_8		r_9	$obligation$	m_8	1 week
n_9		r_{10}	$obligation$	m_9	1 week

TABLE 9.4 – Normes de l'application « house_building »

9.2.4 Spécification d'entrée / sortie

La quatrième dimension de l'OS de notre cas d'étude décrit les moyens définis pour la gestion de la propriété d'entrée / sortie de la problématique d'ouverture étudiée dans nos travaux. Ainsi la spécification d'entrée / sortie (EES) comprend la spécification des portails qui permettront aux agents d'accéder aux informations relatives aux procédures d'entrée / sortie. Les exigences d'entrée / sortie de ce cas d'étude ont une particularité liée au domaine –travaux de construction d'édifices– de cette application. En effet, nous considérons qu'un agent qui souhaite entrer dans une instance d'organisation de ce cas d'étude représente une entreprise prestataire de services liées aux travaux de construction d'édifices. Ainsi, nous nous sommes inspirés des procédures réelles d'attribution de marchés dans le domaine de travaux de bâtiment pour définir

les types d'exigences d'entrée / sortie. Pour nos expérimentations, parmi les diverses caractéristiques ou données que doit remplir ou fournir une entreprise candidate à un appel d'offre de travaux de construction, nous avons retenu les suivantes : le chiffre d'affaire de l'entreprise, l'effectif de son personnel, les moyens qu'elle met en oeuvre en terme de démarche de sécurité de ses employés dans l'exercice de leurs tâches et la démarche environnementale qui doit être appliquée par ses employés pendant la réalisation de leurs tâches afin de préserver le plus possible leur environnement de travail.

Les valeurs de ces différentes données dépendent du travail à réaliser décrit dans l'appel d'offre et ne sont pas toujours communiquées lors de l'appel d'offre. La spécification d'entrée sortie de l'application est donc décrite ci-dessous.

- $EES = \langle P \rangle$, $P = \{p_1, p_2, p_3\}$
  - $p_1 = \{Gr_{p1}, Ga_{p1}, EntryReq_{p1}, ExitReq_{p1}, type_{p1}, hasgate_{p1}\}$
    - $Gr_{p1} = \{building\_house, bigworks\_grp\}$
    - $Ga_{p1} = \{ga_0, ga_1, ga_2, ga_2\}$
    - $EntryReq_{p1} = \{enr_1p_1, enr_2p_1, enr_3p_1\}$
      - $enr_1p_1 = \langle EntryReq, Turnover == \_, mand \rangle$
      - $enr_2p_1 = \langle EntryReq, StaffsSize == \_, mand \rangle$
      - $enr_3p_1 = \langle EntryReq, EnvironmentRules == \_, mand \rangle$
    - $ExitReq_{p1} = \phi$
  - $p_2 = \{Gr_{p2}, Ga_{p2}, EntryReq_{p2}, ExitReq_{p2}, type_{p2}, hasgate_{p2}\}$
    - $Gr_{p2} = \{fitting\_grp\}$
    - $Ga_{p2} = \{ga_4, ga_5, ga_6\}$
    - $EntryReq_{p2} = \{enr_1p_2, enr_2p_2, enr_3p_2\}$
      - $enr_1p_2 = \langle EntryReq, Turnover == \_, mand \rangle$
      - $enr_2p_2 = \langle EntryReq, StaffsSize \leq v_1 \wedge StaffsSize \geq v_2, mand \rangle$ $v_1, v_2 \in \mathbb{N}$
      - $enr_3p_2 = \langle EntryReq, EnvironmentRules == \_, mand \rangle$
    - $ExitReq_{p2} = \phi$
  - $p_3 = \{Gr_{p3}, Ga_{p3}, EntryReq_{p3}, ExitReq_{p3}, type_{p3}, hasgate_{p3}\}$
    - $Gr_{p3} = \{finition\_grp\}$
    - $Ga_{p3} = \{ga_7, ga_8, ga_9\}$
    - $EntryReq_{p_3} = \{enr_1p_3, enr_2p_3, enr_3p_3\}$
      - $enr_1p_3 = \langle EntryReq, Turnover \leq v_1 \wedge Turnover \geq v_2, mand \rangle$ $v_1, v_2 \in \mathbb{N}$
      - $enr_2p_3 = \langle EntryReq, StaffsSize == \_, mand \rangle$

– $enr_3p_3 = \langle EntryReq, EnvironmentRules == \_, mand \rangle$

– $ExitReq_{p3} = \phi$

$gr \in Gr_{p1} \cup Gr_{p2} \cup Gr_{p3}$	$type_p(gr)$	$hasgate_p(gr)$
$building\_house$	EntryExit	$\{ga_0\}$
$bigworks\_grp$	EntryExit	$\{ga_1, ga_2, ga_2\}$
$fitting\_grp$	EntryExit	$\{ga_4, ga_5, ga_6\}$
$finition\_grp$	EntryExit	$\{ga_7, ga_8, ga_9\}$

TABLE 9.5 – Fonctions $type, hasgate$ pour les groupes associés aux portails

$ga \in Ga_{p1} \cup Ga_{p2} \cup Ga_{p3}$	**hastarget**$_{ga}$
ga_0	$(house\_owner, \_, \_)$
ga_1	$(site\_prep\_contractor, \_, \_)$
ga_2	$(bricklayer, \_, \_)$
ga_3	$(roofer, \_, \_)$
ga_4	$(plasterer, \_, \_)$
ga_5	$(plumber, \_, \_)$
ga_6	$(electrician, \_, \_)$
ga_7	$(joiner, \_, \_)$
ga_8	$(tiles\_layer, m_8, \_)$
ga_9	$(frontage\_maker, m_9, g_{9a})$

TABLE 9.6 – Fonctions $hastarget$ des portes d'entrée

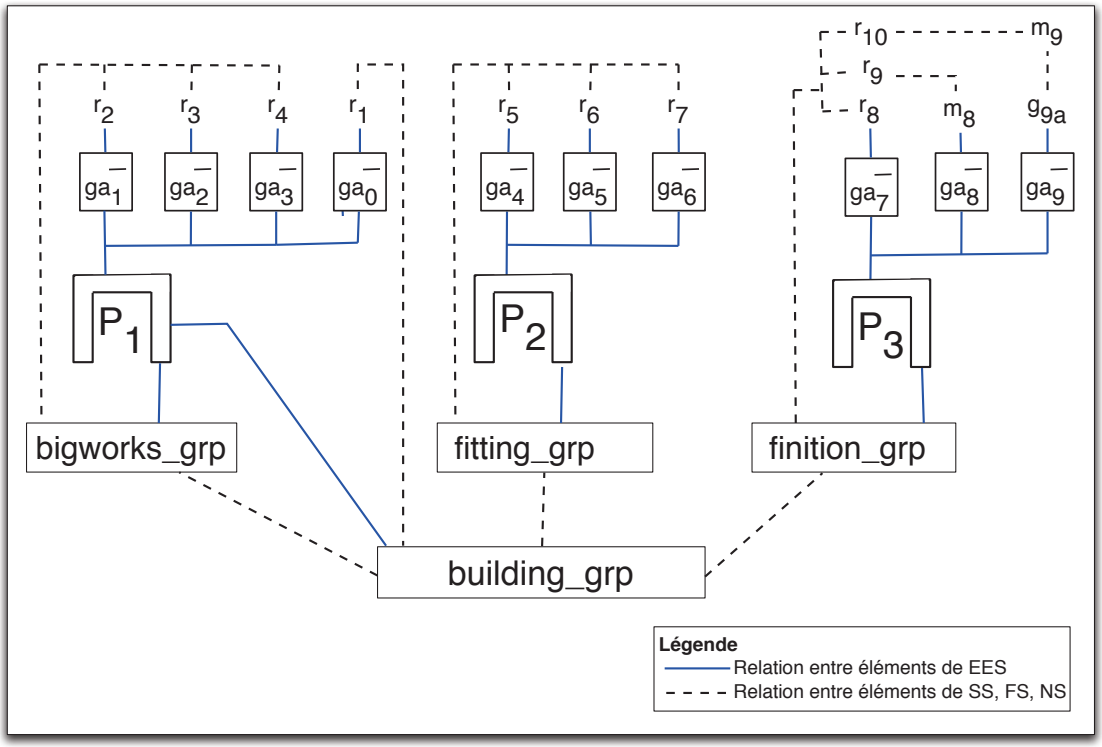


FIGURE 9.3 – Représentation de l'EES de l'application « house_building »

9.3 Gestion de l'ouverture d'une entité organisationnelle de « House_building »

Les expérimentations de gestion de l'ouverture au sein d'une entité organisationnelle ayant l'OS de notre cas d'étude décrit ci-dessus sont présentés dans les sections suivantes.

9.3.1 Création des artefacts organisationnels de l'organisation

Dans cette étape, des agents organisationnels créent les artefacts organisationnels qui gèreront les instances de portails, groupes et schémas sociaux qui seront utilisés dans l'entité organisationnelle.

Par défaut, nous considérons que l'agent qui crée un artefact organisationnel est un agent organisationnel et qu'il supervise l'utilisation de cet OrgArt. Il reçoit donc tous les évènements générés par cet OrgArt ainsi que les modifications des propriétés observables de l'OrgArt. De plus, nous avons choisi qu'un agent qui crée une instance de PortalBoard crée également les appels à candidatures de ce PortalBoard.

La création des artefacts organisationnels se déroule de la façon suivante :

1. La création d'un OrgBoard `buildHouse_OrgB` par un agent nommé *hercule*, et la création d'un GroupBoard `buildHouse_buildingGrpB` avec la spécification du groupe building_grp présentée dans la section 9.2.4.

2. La création par un agent nommé *benedicte* d'un GroupBoard `buildHouse_bigworksGrpB` dont la spécification sera celle du groupe bigworks_grp présentée dans la section 9.1. La création d'un PortalBoard `buildHouse_p1PorB` avec la spécification du portail $p1$ présentée dans la section 9.2.4. La création des appels à candidatures `cfac0`, `cfac1`, `cfac2` et `cfac3` respectivement pour les portes `ga0`, `ga1`, `ga2`, `ga3` définies dans la spécification du portail p1.

3. La création d'un GroupBoard `buildHouse_fittingGrpB` avec la spécification du groupe fitting_grp par un agent nommé *kirikou*. La création PortalBoard `buildHouse_p2PorB` avec la spécification du portail $p2$. La création des appels à candidatures `cfac4`, `cfac5` et `cfac6` respectivement pour les portes `ga4`, `ga5` et `ga6` définies dans la spécification du portail p2.

4. La création d'un GroupBoard `buildHouse_finitionGrpB` avec la spécification du groupe finition_grp par un agent nommé *louise*. La création d'un PortalBoard `buildHouse_p3PorB` avec la spécification du portail $p3$. La création des appels à candidatures `cfac7`, `cfac8` et `cfac9` respectivement pour les portes `ga7`, `ga8` et `ga9` définies dans la spécification du portail p3.

5. La création de SchemeBoard `buildHouse_SchB` avec la spécification du schéma social présentée dans la section 9.2 par l'agent *hercule*.

L'illustration des résultats de cette étape est présentée dans l'annexe E.1.

9.3.2 Gestion des procédures d'entrées

Dans cette étape, différents agents proposent leurs candidatures pour les appels à candidatures créés. Les candidatures sont analysées sur la base des exigences spécifiées. Les agents retenus sont ceux dont la candidature répond à toutes les exigences obligatoires c'est-à-dire de type $mandatory$ et dont le coût des buts est le moins disant c'est-à-dire le moins cher par rapport à celui des autres agents candidats pour le même appel à candidature.

La liste ci-dessous présente le nom des agents ayant présenté une candidature pour chaque CFAC suivi du (ou des) nom(s) de celui (ou de ceux) dont la candidature a été retenue.

- `cfac0` : une seule candidature d'un agent *jean*. Celui-ci est admis.
- `cfac1` : trois candidatures des agents *coly, ruth* et *tomy*. Les agents *coly* et *ruth* sont admis.
- `cfac2` : deux candidatures des agents *moly* et *ruth*. L'agent *ruth* est admis.
- `cfac3` : trois candidatures des agents *brico*, *casto* et *leroy*. Les agents *casto* et *leroy* sont admis.
- `cfac4` : cinq candidatures des agents *alix, berthe, romy, popy* et *sean*. Les agents *berthe* et *sean* sont admis.
- `cfac5` : deux candidatures des agents *aya* et *rita*. L'agent *rita* est admis.
- `cfac6` : deux candidatures des agents *bily* et *mily*. L'agent *mily* est admis.
- `cfac7` : deux candidatures des agents *gyde* et *gery*. L'agent *gyde* est admis.
- `cfac8` : une seule candidature d'un agent *alix*. Celui-ci est admis.
- `cfac9` : deux candidatures des agents *lyne* et *jody*. L'agent *lyne* est admis.

Négociation de clause de contrat

Après la gestion des candidatures, nous avons simulé des demandes de modification des clauses de certains agents admis.

- Les agents *berthe* et *sean* ont été admis pour `cfac4` dont la cible de la porte est $(r_5, \_, \_)$. D'après la spécification normative, le rôle r_5 est associée à une seule norme n_4 dont la mission est m_4 qui comprend les buts $g_{4a}, g_{4b}, g_{4c}, g_{4d}$.
- L'agent *sean* envoie un message à l'agent organisationnel *kirikou* qui supervise `buildHouse_p1PorB` pour lui demander de supprimer la clauses relatives aux buts g_{4a}, g_{4c}, g_{4d}. L'agent organisationnel accepte et il supprime les clauses concernées du contrat de *sean* en exécutant l'opération `updateContract` du PortalBoard `buildHouse_p1PorB`.
- L'agent organisationnel propose à l'agent *berthe* d'avoir une seule clause dans son contrat. Celle-ci est associé au but g_{4b}. L'agent *berthe* accepte et les autres clauses relatives à l'appel à candidature `cfac4` sont supprimées dans son contrat.
- L'agent *ruth* a été admis pour `cfac4`. Après la création de son contrat il a accepté toutes les clauses de celui-ci. Cependant après l'enregistrement de son contrat dans le GroupBoard `buildHouse_bigworksGrpB`, il a demandé à l'agent organisationnel *benedicte* qui supervise `buildHouse_bigworksGrpB` de supprimer la clause concernant le but g_{1a}. L'agent organisationnel a accepté la clause a été supprimée dans le contrat de l'agent *ruth* par l'éxécution de l'opération `deleteClause` du GroupBoard `buildHouse_bigworksGrpB`.

L'illustration des résultats de cette étape est présentée dans l'annexe E.2.

9.3.3 Gestion des engagements aux missions et de la régulation

Afin, de pouvoir participer concrètement à la réalisation des buts du schéma social, les agents doivent s'engager sur le SchemeBoard `buildHouse_SchB` aux missions associées aux clauses de leur contrat. Comme nous l'avons vu dans l'exemple présenté dans la section 8.3.1, les agents sont autonomes et peuvent à cette étape du cycle de vie d'une entité organisationnelle s'engager à des missions qui ne sont associées à aucune clause de leur contrat. Ces comportements sont alors qualifiés de transgression de norme et sont traités par les agents organisationnels selon des stratégies de régulations. Dans nos expérimentations avec ce cas d'étude, nous n'avons pas fait de simulation de transgression de norme. L'illustration de cette étape présentée dans l'annexe E.3 montre essentiellement les résultats des engagements aux missions.

9.3.4 Gestion des coopérations à la réalisation des buts

La coopération à la réalisation des buts commence lorsque le nombre des agents engagés à chaque mission de `buildHouse_SchB` est égal à la cardinalité minimale de la mission. Elle consiste pour les agents engagés aux différentes missions de réaliser les buts correspondants aux clauses de leurs contrats et donc aux missions auxquelles ils se sont engagés dans le schéma social dès que celles-ci seront à l'état *possible*.

9.4 Synthèse

Dans ce chapitre, nous avons présenté le cas d'étude utilisé pour illustrer les propositions faites dans cette thèse pour la gestion de l'ouverture au sein d'organisations multi-agents. Les quatre dimensions structurelle, fonctionnelle, normative et entrée/sortie de ce cas d'étude nous ont permis de mieux présenter les différents concepts proposés dans cette thèse. Ainsi, la spécification des exigences d'adoption et d'abandon dans la SS et la FS de la spécification organisationnelle de ce cas d'étude nous a permis de montrer qu'il est possible de définir différents types d'exigences pour la gestion des adoptions de rôles, missions et buts au cours des processus d'entrée et de sortie et en association avec les exigences d'entrée/sortie. Par ailleurs, l'EES de ce cas d'étude illustre également la richesse de nos propositions avec la possibilité de mutualiser la gestion des entrées dans différents groupes auprès d'un même portail. De plus les différents moyens d'entrée dans une organisation sont également illustrés dans l'EES de ce cas d'étude avec le $portal_3$ dans lequel on a trois portes qui sont respectivement de la forme $(r, \_, \_)$, $(r, m, \_)$ et (r, m, g).

Les expérimentations que nous avons réalisées et présentées dans l'annexe E nous ont permis de montrer quelques fonctionnalités de gestion d'une entité organisationnelle ouverte. Ainsi, la gestion n'est plus centralisé mais distribuée auprès de plusieurs instances d'artefacts organisationnelles. Pour notre cas d'étude, nous avons créé un OrgBoard, quatre GroupBoard, un SchemeBoard et trois PortalBoard. Ces instances d'artefacts organisationnels se coordonnent pour assurer la cohérence de l'entité organisationnelle. Ils sont supervisés par des agents organisationnels qui mettent ainsi en oeuvre la décentralisation des décisions au sein d'une entité organisationnelle ouverte.

Ces expérimentations ont également permis de montrer la gestion des processus d'entrées pour divers appels à candidatures correspondant aux différentes portes définies dans les portails de l'EES. Nous avons ainsi pu voir différents contrats créés

après l'admission d'agents de domaines au sein d'une entité organisationnelle. Á la suite de la création des contrats nous avons montré comment les agents s'engagent progressivement aux missions concernées par les clauses de leur contrat. Les engagements des agents aux missions leur permettent d'initialiser leurs coopérations à la réalisation de l'objectif global de l'entité organisationnelle qui est la construction d'un édifice. La suite de la coopération se ferra par la réalisation des différents buts des différentes clauses des contrats.

Ces expérimentations nous ont également permis de voir quelques limites de nos travaux en particulier en ce qui concerne le passage à l'échelle qui limite le nombre d'agents pouvant être supporté par l'OMI ORA4MAS. Cette limite ne nous a pas permis d'illustrer la coopération à la réalisation des buts du schéma social de notre cas d'étude avec tous les agents de notre scénario d'expérimentation présentés dans la section 9.3.2. En conséquence, nous envisageons de refaire le scénario d'expérimentation de façon à réduire le nombre d'agents pour avoir un nombre maximum pouvant être supporté par l'OMI ORA4MAS et permettant d'illustrer toutes les fonctionnalités de gestion de l'ouverture au sein d'organisations multi-agents. Ensuite, nous envisageons de réaliser des travaux permettant d'améliorer les performances de notre OMI afin de lui permettre de supporter un plus grand nombre d'agents et ainsi de permettre son passage à l'échelle. Par ailleurs, nous envisageons également de réaliser des expérimentations de gestion de processus de sortie que nous avons proposés dans cette thèse. Quant à la gestion du contrôle, l'approche proposée dans cette thèse a besoins d'être améliorée avec une meilleur spécification des fonctions des agents organisationnels et l'implémentation de mécanismes permettant la mise en oeuvre de ces fonctions.

Quatrième partie

Conclusion

Chapitre 10

Conclusion

Nous avons présenté dans les parties 1, 2 et 3 de ce manuscrit, les travaux que nous avons réalisés dans cette thèse. Le bilan ci-dessous résumera les contributions ainsi que les perspectives de recherche pour l'amélioration et l'extension de ces travaux.

Rappel du contexte

Notre objectif dans cette thèse a été d'établir les besoins liés à la problématique d'ouverture dans les organisations multi-agents et de proposer un modèle de spécification et de gestion qui assure la mise en place de l'ouverture au sein d'organisations multi-agents.

L'état de l'art nous a permis de donner une définition d'un système informatique ouvert comme un système dans lequel les interactions peuvent impliquer des entités (ressources matérielles, logicielles ou humaines) qui n'appartiennent pas au système. On dit aussi que ces entités sont dans l'environnement externe du système [71, 134]. Les interactions entre un système et des entités de son environnement externe consistent généralement en des échanges de données. Elles peuvent aussi entraîner des changements d'état du système ainsi que l'ajout ou la suppression d'entité dans le système. Nous avons particularisé la définition d'un système informatique ouvert aux systèmes multi-agents ainsi : un système multi-agent ouvert est un système dans lequel des agents hétérogènes en particulier du point de vue de leurs langages de programmation peuvent entrer, coopérer et sortir tout en conservant leur autonomie. Enfin, nous avons considéré qu'une organisation multi-agent ouverte, est une organisation dans laquelle des agents hétérogènes en termes de langages de programmation, d'architectures de conception ou de modes de raisonnement peuvent entrer, partici-

per à la réalisation d'un objectif global ou collectif suivant des schémas sociaux et des normes prédéfinies et en sortir tout en conservant tout ou partie de leur autonomie.

Nous avons ainsi mis en avant trois propriétés essentielles qui caractérisent la problématique de l'ouverture au sein d'organisations multi-agents.

- La *gestion de l'interopérabilité* : elle consiste pour une organisation multi-agent ouverte à mettre à la disposition des agents hétérogènes, internes et externes à son environnement les informations et les moyens leur permettant d'interagir avec l'organisation.
- La *gestion des entrées et des sorties des agents* : elle consiste pour une organisation multi-agent ouverte à offrir des moyens permettant aux agents hétérogènes de pouvoir entrer et sortir de l'organisation. Ces moyens doivent permettre d'exprimer les processus d'entrée / sortie relativement à la structure de l'organisation et à son objectif global afin de permettre l'intégration appropriée des agents qui entrent et la gestion de l'adaptation de l'organisation après la sortie des agents.
- La *gestion du contrôle des comportements des agents relativement aux normes de l'organisation* : elle consiste à mettre en oeuvre des stratégies de régulations qui préservent tout ou partie de l'autonomie des agents hétérogènes, et qui veillent à la réalisation de l'objectif global de l'organisation.

Nous avons répondu à ces différentes propriétés, par nos contributions que nous résumons ci-dessous.

10.1 Contributions

Les contributions de nos travaux se situent au niveaux de la représentation explicite d'une organisation ouverte avec un langage de modélisation d'organisation (OML) et la gestion du cycle de vie d'une organisation ouverte avec une infrastructure de gestion d'organisation (OMI).

Contributions relatives à l'OML

Le langage de modélisation d'organisation MOISE présenté dans ce mémoire apporte une solution pertinente à la représentation des principales dimensions d'une organisation à savoir structurelle, fonctionnelle, normative ainsi qu'à celle de ses processus d'entrée et de sortie. La dimension structurelle permet de spécifier les groupes,

les rôles d'une organisation. La dimension fonctionnelle permet de spécifier les schémas sociaux de l'organisation et pour chaque schéma social ses buts et ses missions à réaliser ; chaque mission étant composé d'un sous-ensemble de buts. La dimension normative permet de spécifier les normes qui associent les rôles et les missions d'une organisation. La dimension d'entrée / sortie permet de spécifier les portails d'entrée / sortie d'une organisation et pour chaque portail ses exigences d'entrée, ses exigences de sortie et ses portes qui sont les moyens d'entrée dans une organisation.

Il constitue une solution à la propriété d'interopérabilité pour les organisations multi-agents ouvertes dans la mesure où, la spécification organisationnelle (OS) qui n'est autre que l'ensemble des représentations des différentes dimensions d'une organisation citées ci-dessus est une information qui est mise à la disposition des agents. Ainsi, les agents extérieurs à une organisation peuvent accéder à son OS afin de connaître ses spécifications structurelle (SS), fonctionnelle (FS), normative (NS) et d'entrée / sortie (EES).

L'OML MOISE répond également au besoin de la propriété d'entrée / sortie en offrant des moyens de définir de façon explicite des processus d'entrée et de sortie des agents à travers la spécification d'entrée / sortie (EES). En particulier, l'approche proposée offre une flexibilité pour la spécification des moyens d'entrée dans une organisation avec trois types de porte d'entrée dont les représentations respectives correspondent à : (1) la spécification d'un *rôle*, d'une *mission* précise et d'un *but* précis à réaliser. (2) La spécification d'un *rôle* et d'une *mission* précise sans spécification de but précis à réaliser. Les agents admis pourront négocier les buts qu'ils souhaitent réalisés parmi ceux qui appartiennent à la mission. (3) La spécification d'un *rôle* sans spécification de mission ni de but précis à réaliser. Les agents admis pourront négocier les missions auxquelles ils souhaitent s'engager parmi les missions associées au rôle par les normes de la NS.

Les normes définies dans la spécification normative (NS) d'une organisation décrivent les comportements escomptés des agents, et implicitement les règles de contrôle des actions des agents au sein d'une organisation. Elles constituent donc une solution à la propriété de contrôle dans des organisations multi-agents ouvertes.

Contributions relatives à l'OMI

L'infrastructure de gestion d'organisation ORA4MAS basée sur le concept d'*artefact organisationnel* gère des instances d'organisations –que nous appelons *entités organisationnelles* – dont le fonctionnement est régi par une spécification organisationnelle correspondante décrite avec l'OML MOISE. Les artefacts organi-

sationnels sont supervisés par des *agents organisationnels*. Les agents qui entrent dans une entité organisationnelle et qui participent à la réalisation des buts du schéma social global de celle-ci sont appelés *agents de domaines*. Nous avons défini quatre types d'artefacts organisationnels pour l'OML MOISE : OrgBoard, PortalBoard, GroupBoard et SchemeBoard. Ainsi, ORA4MAS assure la gestion distincte des processus d'entrée / sortie avec des instances de PortalBoard, la gestion des contrats des agents avec des instances de GroupBoard, la gestion des coordinations de la réalisation des buts des schémas sociaux et du contrôle des agents avec des instances de SchemeBoard. Cet OMI permet donc d'une part de distribuer la gestion des fonctionnalités de gestion de l'ouverture au sein d'une entité organisationnelle multi-agent avec les artefacts organisationnels. D'autre part, elle permet de décentraliser la gestion des décisions de supervision des interactions des agents de domaines avec des agents organisationnels.

En ce qui concerne les propriétés de gestion de l'interopérabilité, des entrées / sorties et du contrôle d'une entité organisationnelle par les artefacts organisationnels de ORA4MAS, elles sont liées à l'approche conceptuelle d'un artefact [117] notamment avec ses propriétés observables et ses opérations. Ainsi, les propriétés observables des artefacts organisationnels mettent à la disposition des agents de domaines des informations d'une entité organisationnelle (e.g. OS, appels à candidatures, résultats des appels à candidatures, contrats des agents, buts à réaliser, ...). Ces informations leur permettent de raisonner sur leurs interactions (candidature d'entrée, négociation de contrat, coopération à la réalisation des buts, demande de sortie, ...) avec l'entité organisationnelle. Les opérations des artefacts organisationnels, permettent aux agents de domaines d'avoir diverses interactions avec une entité organisationnelle. Par exemple, les démarches d'entrée et de sortie se font principalement par l'exécution d'opérations des artefacts organisationnels. De même, pour les agents dont la démarche d'entrée a réussi, les engagements aux missions des clauses de leurs contrats, et la participation à la réalisation des buts des schémas sociaux de l'entité organisationnelle se font en exécutant des opérations des artefacts organisationnels. L'approche de contrôle proposée et développée dans ORA4MAS comprend trois étapes dont la première est assurée par les artefacts organisationnels. Concrètement, lorsqu'à la suite de l'exécution d'une opération par un agent de domaine on a une transgression de norme, c'est l'artefact organisationnel ayant géré l'exécution de l'opération qui génère les données permettant de constater la transgression de norme. Les deux autres étapes de notre approche de contrôle sont gérées par les agents organisationnels.

Les fonctions de supervision des agents organisationnels, répondent également aux

propriétés de gestion de l'interopérabilités, des entrées / sorties et de contrôle. En effet, ce sont les agents organisationnels qui créent les appels à candidatures que les agents de domaines peuvent consulter au moyen des propriétés observables. La gestion des entrées et des sorties est supervisée par les agents organisationnels qui interviennent particulièrement lors de la négociation des clauses de contrats. Quant au contrôle, les décisions relatives aux régulations des comportements malveillants (transgression des normes) des agents de domaines sont prises par les agents organisationnels.

Enfin, les agents organisationnels et les agents de domaines d'une entité organisationnelles peuvent également échanger des messages que ce soit au cours de processus d'entrée / sortie, ou lors de la gestion du contrôle et en particulier des régulations. Les échanges de messages sont gérées par les API qui implémentent les langages de programmation agent en l'occurrence le langage Jason [22] que nous avons utilisé pour développer les agents pour nos expérimentations.

10.2 Limites et perspectives

La problématique de cette thèse étant assez récente dans le domaine des organisations multi-agents, chaque étape de nos travaux nous a révélé un nombre considérables de limites de nos propositions et donc de perspectives que la communauté, à défaut de votre servante, se doit d'explorer. Ces limites se situent aussi bien au niveau du langage de modélisation qu'au niveau de l'infrastructure de gestion d'organisation.

Limites et perspectives de l'OML

Le langage de modélisation d'organisation MOISE, n'offre pas la possibilité de spécifier l'ontologie du domaine d'application de l'organisation à modéliser. Ainsi, une bonne interprétation par les agents des termes utilisées pour spécifier les buts et les exigences d'une organisation dans une OS n'est pas garantie. Par exemple, dans le cas d'étude que nous avons présenté, nous avons défini des termes tels que *steelframe_built, tiles_layed, ...* pour désigner quelques buts de l'OS et les termes *team_memb_numb, goal_cost, ...* pour définir certaines exigences. Nous supposons que tout agent interprète le termes *steelframe_built, tiles_layed, team_memb_numb et goal_coast* comme désignant respectivement un but de construction des cloisons d'un édifice, un but de pose des carreaux dans un édifice, une exigence qui désigne le nombre de personnel de l'équipe de travail et une exigence qui désigne le coût d'un but que va réaliser un agent. Cependant, nous ne pouvons garantir que tous les agents ont cette interprétation. L'intégration d'une spécification ontologique dans l'OML

MOISE pourra donc servir de garantie d'une bonne interprétation de la sémantique des buts et des exigences définies dans une OS.

Limites et perspectives de l'OMI

En ce qui concerne l'infrastructure de gestion d'organisation ORA4MAS, ses limites concernent les trois propriétés qui caractérisent l'ouverture dans les organisations multi-agents.

Limites et améliorations liées à l'interopérabilité

Les agents que nous avons développés pour nos expérimentations interagissent avec les artefacts avec une hypothèse forte de connaissance des opérations, des évènements ou perceptions générés par les opérations et de la représentation des données des propriétés observables. Il n'a donc pas été possible d'évaluer la capacité pour des agents hétérogènes c'est-à-dire de langages de programmation différents tels que Jade [15], Jadex [4], Jason [22], 2APL [48], ... et n'ayant aucune connaissance a priori sur une entité organisationnelle d'interagir avec elle de façon appropriée. Ainsi, le développement des manuels d'utilisations des artefacts pourra contribuer à l'évaluation de l'interopérabilité de ORA4MAS du point de vue de l'hétérogénéité des agents.

Limites et améliorations liées à la gestion des entrées / sorties

Dans la gestion des processus d'entrée, l'approche d'analyse des candidatures et en particulier des valeurs fournies par les agents pour les exigences des appels à candidatures se limite à la comparaison de valeurs de type simple (numérique et chaîne de caractère). Ainsi, pour les expérimentations réalisées avec le cas d'étude, il n'a pas été possible de tenir compte des exigences de type complexe tel que la description de la démarche écologique (préservation de l'environnement) fournie par les agents. Il sera donc intéressant d'améliorer les algorithmes d'analyse d'exigences afin qu'elles puissent traiter des exigences de type complexe et éventuellement intégrer la prise en compte de l'ontologie de l'organisation.

Par ailleurs, la stratégie d'admissibilité des agents que nous avons développée consiste à sélectionner les candidats qui fournissent au moins toutes les exigences obligatoires. Cette stratégie peut ne pas être appropriée pour toutes les applications. L'intégration d'une spécification de critère d'admissibilité dans la spécification de processus d'entrée offrira une flexibilité non négligeable à la valorisation de nos propositions.

En ce qui concerne la gestion des sorties, nous n'avons pas développé de stratégie d'adaptation permettant de gérer des cas de sortie inopportun des agents. L'intégration d'approches de réorganisation et de création en cours d'exécution d'un schéma social de nouveaux appels à candidatures pour l'attribution de rôles, de missions ou de but vacants permettra d'assurer un bon déroulement de la réalisation des buts de l'organisation.

Enfin, les échanges de messages entre les agents organisationnels et les agents de domaines lors de la négociation de clause de contrat ne sont pas régis par des protocoles bien définis. La spécification des protocoles d'échanges entres agents nous semble donc nécessaire comme amélioration future.

Limites et améliorations liées au contrôle

Les mécanismes de gestion des coopérations à la réalisation des buts des schémas sociaux n'incluent pas la vérification effective de la réalisation des buts par les agents. Ainsi, un agent de domaine peut mettre l'état d'un but à *achieved* alors que concrètement le but n'a pas été réalisé. Il sera donc intéressant de développer des moyens permettant de faire ces vérifications ou d'intégrer des travaux abordant cette problématique [110, 111] afin de pallier cette limite.

Quant à la gestion des régulations, la stratégie que nous avons développée n'est pas assez élaborée. Par exemple, nous n'avons pas prévu de moyen de révocation d'agent en cas de transgression de norme. Il serait donc intéressant de développer des opérations d'artefacts organisationnels qui permettent aux agents organisationnels d'appliquer des stratégies de révocation. De plus, divers travaux sont fait dans les SMA sur la réputation des agents [65, 133]. Ceux-ci peuvent enrichir les stratégies de régulation dans les organisations multi-agents ouvertes qui sont essentiellement basées sur les sanctions précédées parfois d'avertissement comme dans nos travaux. En outre, l'intégration de l'approche NOPL [78] pourra également permettre d'améliorer la régulation puisque cette approche permet de définir quelles actions au sein d'une organisation peuvent être ou non transgressées par les agents. De même que pour la gestion des entrées / sorties, les échanges de messages entre les agents organisationnels et les agents de domaines dans des cas de régulation ne sont pas régis par des protocoles bien définis. C'est donc une amélioration nécessaire à faire pour améliorer la gestion des contrôles au sein d'une organisation multi-agent ouverte.

Autres perspectives

En plus des limites ci-dessus cités relatives aux propriétés de l'ouverture, d'autres travaux pourront étendre les contributions de cette thèse à d'autres problématiques. Ainsi, les pistes de travail suivants pourront être explorées :

– Évaluation du passage à l'échelle de ORA4MAS. Les expérimentations que nous avons réalisées avec le cas d'étude n'ont nécessités qu'un petit nombre d'agents. Il serait intéressant de savoir quelle est la robustesse de cette API en nombres d'agents de domaines et d'agents organisationnels qui peuvent interagir avec une entité organisationnelle sans altérer son bon fonctionnement. De plus il serait intéressant d'évaluer les nombres minimum et maximum d'agents organisationnels pouvant être affectés à un artefact organisationnel en fonction du nombre maximum d'agents de domaine susceptibles d'interagir avec l'artefact organisationnel et entrainant des interventions de régulation.

– Intégration de mécanismes de sécurité. Les opérations dédiées aux agents organisationnels comme la création et mise à jour de contrat ne sont pas sécurisées. De plus, certaines propriétés observables telles que les contrats ne devraient être accessibles qu'aux agents organisationnels. La mise en place d'une stratégie de sécurité avec l'approche RBAC peut être une piste de travail à explorer.

– Confrontation de stratégie d'entrée / sortie des agents de domaines à celle d'une organisation. Nous avons essentiellement abordée la spécification et la gestion des entrées / sorties par une organisation. Il serait intéressant de confronter nos propositions de processus d'entrée avec des propositions de gestions de raisonnement d'agents [29, 28] sur leurs compétences, désirs et intentions qui leurs permettent de décider de répondre à un appel à candidature ou d'effectuer une demande de sortie lorsqu'ils sont dans une organisation.

– Étude de critères d'évaluations des infrastructures de gestion d'organisation multi-agents ouvertes. La communauté de modélisation et de gestion d'organisation multi-agents n'ayant pas de critère commun d'évaluation, il sera intéressant d'aborder cette problématique afin de pouvoir évaluer notre proposition d'OMI à d'autres OMI existantes ou futures.

Bibliographie

[1] Wikipedia, June 2010. http ://fr.wikipedia.org/.

[2] Wordnet, June 2010. http ://wordnetweb.princeton.edu/.

[3] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, June 2003.

[4] Winfried Lamersdorf Alexander Pokahr, Lars Braubach. Jadex : A BDI reasoning engine. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming : Languages, Platforms, and Applications*, chapter 6. Springer, 2005.

[5] OSGi Alliance. Osgi : Open services gateway initiative. http ://www.osgi.org/. last access Novembre 2011.

[6] Estefania Argente, Vicente J. Botti, C. Carrascosa, A. Giret, Vicente Julián, and M. Rebollo. An abstract architecture for virtual organizations : The thomas project, 2008. Technical Report, http ://users.dsic.upv.es/grupos/ia/sma/tools/Thomas/archivos/arquitectura.pdf.

[7] Estefania Argente, Javier Palanca Cámara, Gustavo Aranda Bada, Vicente Julián, Vicente J. Botti, Ana García-Fornes, and Agustín Espinosa. Supporting agent organizations. In Hans-Dieter Burkhard, Gabriela Lindemann, Rineke Verbrugge, and László Zsolt Varga, editors, *5th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'07)*, volume 4696 of *Lecture Notes in Computer Science*, pages 236–245. Springer, 2007.

[8] Alexander Artikis and Jeremy Pitt. Specifying open agent systems : A survey. In Alexander Artikis, Gauthier Picard, and Laurent Vercouter, editors, *"Engineering Societies in the Agents World"(ESAW'08)*, volume 5485 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2009.

[9] B. Moulin B. Chaib-draa and I. Jarras. Agent et systèmes multiagents. In JP. Briot et Y Demazeau, editor, *Principes et architecture des systèmes multi-agent*. Hermes Lavoisier, 2001.

[10] Bala M. Balachandran and Majigsuren Enkhsaikhan. Developing multi-agent e-commerce applications with jade. In Bruno Apolloni, Robert J. Howlett, and Lakhmi C. Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems, 11th International Conference (KES 3)*, volume 4694 of *Lecture Notes in Computer Science*, pages 941–949. Springer, 2007.

[11] Edward A. Roback barbara Guttman. An introduction to computer security : The nist handbook. Technical report, Washington, 1995.

[12] K. Suzanne Barber, Karen Fullam, and Joonoo Kim. Challenges for trust, fraud and deception research in multi-agent systems. In Rino Falcone, K. Suzanne Barber, Larry Korba, and Munindar P. Singh, editors, *Trust, Reputation, and Security : Theories and Practice, AAMAS 2002*, volume 2631 of *Lecture Notes in Computer Science*, pages 8–14. Springer, 2002.

[13] K. Suzanne Barber and Dung N. Lam. Specifying and analyzing agent architectures using the agent competency framework. In *Fifteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2003)*, pages 232–239, 2003.

[14] K. Suzanne Barber and Jisun Park. Agent belief autonomy in open multi-agent systems. In Matthias Nickles, Michael Rovatsos, and Gerhard Weiß, editors, *Agents and Computational Autonomy*, volume 2969 of *Lecture Notes in Computer Science*, pages 7–16. Springer, 2003.

[15] Fabio Bellifemine, Giovanni Caire, Agostino Poggi, and Giovanni Rimassa. *JADE : A software framework for developing multi-agent applications. Lessons learned*, volume 50. 2008.

[16] Carole Bernon, Vincent Chevrier, Vincent Hilaire, and Paul Marrow. Applications of self-organising multi-agent systems : An initial framework for comparison. *Informatica (Slovenia)*, 30(1) :73–82, 2006.

[17] Philippe Bernoux. *La sociologie des organisations*. Points, 1987.

[18] Guido Boella, Pablo Noriega, Gabriella Pigozzi, and Harko Verhagen, editors. *Normative Multi-Agent Systems*. Number 09121 in Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, Dagstuhl, Germany, 2009.

[19] Guido Boella, Leendert W. N. van der Torre, and Harko Verhagen. Introduction to normative multiagent systems. *Computational and Mathematical Organization Theory*, 12(2-3) :71–79, 2006.

[20] Olivier Boissier, Jomi Fred Hübner, and Jaime Simão Sichman. Organization oriented programming from closed to open organizations. In Gregory O'Hare, Michael O'Grady, Oguz Dikenelli, and Alessandro Ricci, editors, *Engineering Societies in the Agents World VII*, volume 4457 of *LNCS*. Springer-Verlag, 2007.

[21] Olivier Boissier, Rosine Kitio, Jomi Fred Hübner, and Alessandro Ricci. Instrumentation d'organisations multi-agents avec des artefact organisationnels. In René Mandiau and Pierre Chevaillier, editors, *Actes des 16e Journées Francophones sur les Systèmes Multi-Agents (JFSMA'08)*, pages 107–116. Cépaduès, 2008.

[22] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldrige. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. John Wiley & Sons, 2007.

[23] Jeffrey M. Bradshaw, Stewart Dutfield, Pete Benoit, and John D. Woolley. Kaos : toward an industrial-strength open agent architecture. pages 375–418, 1997.

[24] Giacomo Cabri, Luca Ferrari, and Letizia Leonardi. Enabling mobile agents to dynamically assume roles. In *SAC '03 : Proceedings of the 2003 ACM symposium on Applied computing*, pages 56–60, New York, NY, USA, 2003. ACM.

[25] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Brain : A framework for flexible role-based interactions in multiagent systems. In Robert Meersman, Zahir Tari, and Douglas C. Schmidt, editors, *CoopIS/DOA/ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2003.

[26] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Implementing role-based interactions for internet agents. In *SAINT*, pages 380–389. IEEE Computer Society, 2003.

[27] Lui M. Camarinha-Matos and Hamideh Afsarmanesh. Virtual enterprise modeling and support infrastructures : applying multi-agent system approaches. pages 335–364, 2001.

[28] Cosmin Carabelea. *Raisonner sur l'autonomie d'un agent au sein de systèmes multi-agents ouverts*. PhD thesis, Ecole Nationale Supèrieure des Mines, Saint Etienne, 2007.

[29] Cosmin Carabelea and Olivier Boissier. Coordinating agents in organizations using social commitments. *Electr. Notes Theor. Comput. Sci.*, 150(3) :73–91, 2006.

[30] Cosmin Carabella. *Raisonner sur l'autonomie d'un agent au sein de systèmes mutli-agents ouverts : *une approche basée sur les relations de pouvoir*. PhD thesis, Ecole Nationale Supèrieure des Mines, Saint Etienne, 2007.

[31] Henrique Lopes Cardoso and Eugénio Oliveira. Flexible deadlines for directed obligations in agent-based business contracts. In *AAMAS '09 : Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 1307–1308, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[32] Henrique Lopes Cardoso and Eugénio C. Oliveira. Virtual enterprise normative framework within electronic institutions. In Marie Pierre Gleizes, Andrea Omicini, and Franco Zambonelli, editors, *ESAW*, volume 3451 of *Lecture Notes in Computer Science*, pages 14–32. Springer, 2004.

[33] Henrique Lopes Cardoso and Eugénio C. Oliveira. A contract model for electronic institutions. In Jaime Simão Sichman, Julian A. Padget, Sascha Ossowski, and Pablo Noriega, editors, *COIN*, volume 4870 of *Lecture Notes in Computer Science*, pages 27–40. Springer, 2007.

[34] Henrique Lopes Cardoso and Eugénio C. Oliveira. Monitoring directed obligations with flexible deadlines : A rule-based approach. In Matteo Baldoni, Jamal Bentahar, M. Birna van Riemsdijk, and John Lloyd, editors, *Declarative Agent Languages and Technologies, DALT'09*, volume 5948 of *Lecture Notes in Computer Science*, pages 51–67. Springer, 2009.

[35] David Chen and Guy Doumeingts. European initiatives to develop interoperability of enterprise applications –basic concepts, framework and roadmap. *Annual Reviews in Control*, 27 :153–162, 2003.

[36] Philip R. Cohen, Adam Cheyer, Michelle Wang, and Soon Cheol Baeg. An open agent architecture. pages 197–204, 1998.

[37] COIN. Coordination, organization, institutions and norms in agent systems (coin) : The international workshop series. http ://www.pcs.usp.br/ coin/. last access Novembre 2011.

[38] Alan W. Colman and Jun Han. Operational management contracts for adaptive software organisation. In *Australian Software Engineering Conference*, pages 170–179. IEEE Computer Society, 2005.

[39] Alan W. Colman and Jun Han. Organizational roles and players. In *Fall Symposium on Roles an interdisciplinary perspective*, pages 55–62, 2005.

[40] Dublin Core. Interoperability levels. http ://dublincore.org/documents/interoperability-levels/, June 2010. last access November 2011.

[41] Luciano R. Coutinho, Anarosa Brandão, Jaime Simão Sichman, and Olivier Boissier. Model-driven integration of organizational models. In Michael Luck and Jorge J. Gómez-Sanz, editors, *AOSE*, volume 5386 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008.

[42] Luciano R. Coutinho, Jaime S. Sichman, and Olivier Boissier. Modeling organization in mas : A comparison of models. In *In SEAS 05 : 1st Workshop on Software Engineering for Agent-oriented Systems*, 2005.

[43] Luciano R. Coutinho, Jaime S. Sichman, and Olivier Boissier. *Handbook of Research on Multi-Agent Systems : Semantics and Dynamics of Organizational Models*, chapter Modelling Dimensions for Agent Organizations, pages 2–18. Information Science Reference Publisher, 2009. ISBN 978-1-60566-256-5.

[44] Ed Coyne and Tim Weil. Role-based access control implementation standard. InterNational Committee for Information Technology Standards (INCITS), 2007. Proposed standard ; InterNational Committee for Information Technology Standards.

[45] Ed Coyne and Tim Weil. An rbac implementation and interoperability standard : The incits cyber security 1.1 model. *IEEE Security and Privacy*, 6(1) :84–87, 2008.

[46] Frédéric Cuppens and Alexandre Miège. Administration Model for Or-BAC. In *Computer Systems Science and Engineering (CSSE'04)*, volume 19, May, 2004.

[47] Viviane Torres da Silva, Ricardo Choren, and Carlos J. P. de Lucena. A uml based approach for modeling and implementing multi-agent systems. *Autonomous Agents and Multiagent Systems, International Joint Conference on*, 2 :914–921, 2004.

[48] Mehdi Dastani. 2apl : a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3) :214–248, 2008.

[49] Mehdi Dastani, Virginia Dignum, and Frank Dignum. Role-assignment in open agent societies. In *AAMAS*, pages 489–496. ACM, 2003.

[50] Virginia Dignum and Frank Dignum. Towards an agent-based infrastructure to support virtual organisations. In *PRO-VE '02 : Proceedings of the IFIP TC5/WG5.5 Third Working Conference on Infrastructures for Virtual Enterprises*, pages 363–370, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V.

[51] Virginia Dignum and Frank Dignum. What's in it for me ? agent deliberation on taking up social roles. In *EUMAS*, 2004.

[52] Virginia Dignum, John-Jules Meyer, and Hans Weigand. Towards an organizational model for agent societies using contracts. In *AAMAS '02 : Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 694–695, New York, NY, USA, 2002. ACM.

[53] Edmund H. Durfee. The distributed artificial intelligence melting pot. *IEEE Transactions on Systems, Man, and Cybernetics*, 21 :1301–1306, 1991.

[54] Amal El Fallah Seghrouchni, Serge Haddad, Tarek Melliti, and Alexandru Suna. Interopérabilité des systèmes multi-agents à l'aide des services web. In Boissier Olivier and Zahia Guessoum, editors, *Actes des 12èmes Journées Francophones des Systèmes Multi-Agents (JFSMA'04)*, pages 91–104, Paris, France, November 2004. Hermès.

[55] Göktürk Erek, Ay¸se Kiper, Ismail, Hakk Toroslu, and Faruk Polat. Implementing kqml agent communication language for multiagent simulation architectures on hla, 2003.

[56] Marc Esteva, David De La Cruz, and Carles Sierra. Islander : an electronic institutions editor. In *AAMAS 02 : Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1045–1052, New York, NY, USA, 2002. ACM Press.

[57] Marc Esteva, Juan A. Rodríguez-Aguilar, Bruno Rosell, and Josep L. AMELI : An agent-based middleware for electronic institutions. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Milind Tambe, editors, *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 236–243, New York, 2004. ACM.

[58] Rino Falcone, K. Suzanne Barber, Larry Korba, and Munindar P. Singh, editors. *Trust, Reputation, and Security : Theories and Practice, AAMAS 2002 International Workshop, Bologna, Italy, July 15, 2002, Selected and Invited Papers*, volume 2631 of *Lecture Notes in Computer Science*. Springer, 2003.

[59] Rino Falcone, K. Suzanne Barber, Jordi Sabater-Mir, and Munindar P. Singh, editors. *Trust in Agent Societies, 11th International Workshop, TRUST 2008, Estoril, Portugal, May 12-13, 2008. Revised Selected and Invited Papers*, volume 5396 of *Lecture Notes in Computer Science*. Springer, 2008.

[60] J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations : an organizational view of multiagent systems. In LNCS, editor, *AOSE 03 : 4th Agent-Oriented Software Engineering*, volume 2935 of *LNCS*, 2003.

[61] David Ferraiolo and Richard Kuhn. Role-based access control. In *In 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[62] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3) :224–274, 2001.

[63] Catarina Ferreira da Silva, Lionel Médini, Samer Abdul Ghafour, Patrick Hoffmann, and Parisa Ghodous. Semantic Interoperability of Heterogeneous Semantic Resources. *Electronic Notes in Theoretical Computer Science*, 150(2) :71–85, March 2006. Equipe SICO. http ://dx.doi.org/10.1016/j.entcs.2005.11.03.

[64] FIPA. Foundation for intelligent physical agents (fipa). http ://www.fipa.org. last access Novembre 2011.

[65] Karen K. Fullam, Tomas B. Klos, Guillaume Muller, Jordi Sabater, Andreas Schlosser, Zvi Topol, K. Suzanne Barber, Jeffrey S. Rosenschein, Laurent Vercouter, and Marco Voss. A specification of the agent reputation and trust (art) testbed : experimentation and competition for trust in agent societies. In *AAMAS '05 : Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 512–518, New York, NY, USA, 2005. ACM.

[66] Andrés García-Camino, Juan Antonio Rodríguez-Aguilar, and Wamberto Vasconcelos. A distributed architecture for norm management in multi-agent systems. In *COIN'07 : Proceedings of the 2007 international conference on Coordination, organizations, institutions, and norms in agent systems III*, pages 275–286, Berlin, Heidelberg, 2008. Springer-Verlag.

[67] Les Gasser. Perspectives on organizations in multi-agent systems. pages 1–16, 2001.

[68] Benjamin Gâteau. *Modèlisation et Supervision d'Institution Multi-Agents*. PhD thesis, Ecole Nationale Supèrieure des Mines, Saint Etienne, 2007.

[69] Benjamin Gâteau, Olivier Boissier, and Djamel Khadraoui. Organisation multi-agent normative : modélisation et infrastructure. In Valérie Camps and Philippe Mathieu, editors, *Actes des 15e Journées Francophones sur les Systèmes Multi-Agents (JFSMA'2007)*. Cépaduès, 2007.

[70] Benjamin Gâteau, Olivier Boissier, Djamel Khadraoui, and Eric Dubois. Moiseinst : An organizational model for specifying rights and duties of autonomous agents. In *Third European Workshop on Multi-Agent Systems (EUMAS 2005)*, Brussels Belgium, December 7-8 2005.

[71] John P. Van Gigch. *Applied general systems theory*. HarperCollins-Publishers-Inc, 1974.

[72] Olivier Gutknecht and Jacques Ferber. Madkit : Organizing heterogeneity with groups in a platform for multiple multi-agent systems, 1997. http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.35.7676.

[73] Olivier Gutknecht and Jacques Ferber. Madkit : a generic multi-agent platform. In *AGENTS '00 : Proceedings of the fourth international conference on Autonomous agents*, pages 78–79, New York, NY, USA, 2000. ACM.

[74] Mahdi Hannoun. *MOISE : un modèle organisationnel pour les systèmes multi-agents*. PhD thesis, Université Jean Monnet et Ecole Nationale Supérieure des Mines de Saint-Etienne, 2002.

[75] Carl Hewitt. Offices are open systems. *ACM Transactions on Information Systems*, 4(3) :271–287, 1986.

[76] Jomi F. Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3) :369–400, 2010.

[77] Jomi F. Hübner, Olivier Boissier, and Jaime S. Sichman. Programming MAS reorganisation with MOISE+. In J. Meyer, M. Dastani, and R. Bordini, editors, *Dagstuhl Seminar on Foundations and Practice of Programming Multi-Agent Systems*, volume 06261, 2006.

[78] Jomi Fred Hübner, Olivier Boissier, and Rafael H. Bordini. Normative programming for organisation management infrastructures. In Julian A. Padget, Alexander Artikis, Wamberto Weber Vasconcelos, Kostas Stathis, Viviane Torres da Silva, Eric T. Matson, and Axel Polleres, editors, *Coordination, Organization, Institutions and Norms in Agent Systems (COIN@MALLOW'009)*, volume 494 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[79] Jomi Fred Hubner, Olivier Boissier, and Jaime Simao Sichman. Using a multi-agent organization description language to describe contract dynamics in virtual enterprises.

In *IAT '05 : Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 672–678, Washington, DC, USA, 2005. IEEE Computer Society.

[80] Jomi Fred Hübner, Eric T. Matson, Olivier Boissier, and Virginia Dignum, editors. *Coordination, Organizations, Institutions and Norms in Agent Systems IV, COIN 2008 International Workshops, COIN@AAMAS 2008, Estoril, Portugal, May 12, 2008. COIN@AAAI 2008, Chicago, USA, July 14, 2008. Revised Selected Papers*, volume 5428 of *Lecture Notes in Computer Science*. Springer, 2009.

[81] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. $\mathcal{M}$OISE$^+$: Towards a structural, functional, and deontic model for MAS organization. In Cristiano Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2002)*, pages 501–502. ACM Press, 2002.

[82] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Using the $\mathcal{M}$OISE$^+$ for a cooperative framework of MAS reorganisation. In Ana L. C. Bazzan and Sofiane Labidi, editors, *SBIA*, volume 3171 of *Lecture Notes in Computer Science*. Springer, 2004.

[83] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. $\mathcal{S}$-$\mathcal{M}$OISE$^+$: A middleware for developing organised multi-agent systems. In Olivier Boissier, Virginia Dignum, Eric Matson, and Jaime Simão Sichman, editors, *Proceedings of the International Workshop on Organizations in Multi-Agent Systems, from Organizations to Organization Oriented Programming in MAS (OOOP'2005)*, volume 3913 of *LNCS*. Springer, 2006.

[84] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Developing organised multiagent systems using the moise. *International Journal of Agent-Oriented Software Engineering (IJAOSE)*, 1(3/4) :370–395, 2007.

[85] Bradshaw Jeffrey M., Jung Hyuckchul, Kulkarni Shri, Johnson Matthew, Feltovich Paul, Allen James, Bunch Larry, Chambers Nathanael, Galescu Lucian, Jeffers Renia, Suri Niranjan, Taysom William, and Uszok Andrzej. Kaa :policy-based explorations of a richer model for adjustable autonomy. In *AAMAS '05 : 4th International conference on Autonomous Agents and Multi-Agent Systems*, 2005.

[86] Nicholas R. Jennings, Katia P. Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1) :7–38, 1998.

[87] Martha L. Kahn and Cynthia Della Torre Cicalese. The coabs grid. In Walt Truszkowski, Christopher Rouff, and Michael G. Hinchey, editors, *First International Workshop on Radical Agent Concepts : WRAC'2002*, volume 2564 of *Lecture Notes in Computer Science*, pages 125–134. Springer, 2002.

[88] Rosine Kitio, Olivier Boissier, Jomi Fred Hübner, and Alessandro Ricci. Organisational artifacts and agents for open multi-agent organisations : "giving the power back to the agents". In *Fifth European Workshop on Multi-Agent Systems (EUMAS 2007)*, pages 171–186, 2007. Paper initially published in COIN@MALLOW 2007.

[89] Rosine Kitio, Olivier Boissier, Jomi Fred Hübner, and Alessandro Ricci. Organisational artifacts and agents for open multi-agent organisations : “giving the power back to the agents”. In Jaime Sichman, P. Noriega, J. Padget, and Sascha Ossowski, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems III*, volume 4870 of *LNCS*, pages 171–186. Springer, 2008. Revised Selected Papers.

[90] Yannis Labrou and Tim Finin. A proposal for a new kqml specification, 1997. http ://-citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.38.7779.

[91] Amy L. Lansky. Behavioral specification and planning for multiagent domains, 1985. Technical report : SRI INTERNATIONAL MENLO PARK, http ://-handle.dtic.mil/100.2/ADA461786.

[92] Dictionnaire Larousse. *http ://www.larousse.fr/dictionnaires/francais/*. 2010. Last access Novembre 2011.

[93] Sylvain Lemouzy, Carole Bernon, and Marie Pierre Gleizes. Auto-ajustement de comportements agents par une approche coopérative locale. *Revue d’Intelligence Artificielle*, 23(5-6) :719–748, 2009.

[94] Hector J. Levesque, Philip R. Cohen, and José H. T. Nunes. On acting together. In *AAAI*, pages 94–99, 1990.

[95] Ninghui Li and Ziqing Mao. Administration in role-based access control. In *ASIACCS ’07 : Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 127–138, New York, NY, USA, 2007. ACM.

[96] Robert Longeon and Jean-Luc Archimbaud. Guide de la sécurité des systèmes d’information, 2010. http ://www.dgdr.cnrs.fr/FSD/securite-systemes/documentations_pdf/securite_systemes/guide.pdf.

[97] Thomas W. Malone. Tools for inventing organizations : Toward a handbook of organizational process. *Management Science*, 45(3) :425–443, March 1999.

[98] Philippe Mathieu, Jean-Christophe Routier, and Yann Secq. Rio : Roles, interactions and organizations. In Vladimír Marík, Jörg P. Müller, and Michal Pechoucek, editors, *The 3rd International/Central and Eastern European Conference on Multi-Agent Systems (CEEMAS’03)*, volume 2691 of *Lecture Notes in Computer Science*, pages 147–157. Springer, 2003.

[99] Deborah L. McGuinness, Richard Fikes, James Hendler, and Lynn Andrea Stein. Daml+oil : An ontology language for the semantic web. *IEEE Intelligent Systems*, 17(5) :72–80, 2002.

[100] Tarek Melliti, Serge Haddad, Alexandru Suna, and Amal El Fallah Seghrouchni. Web-MASI : Multi-agent systems interoperability using web services based approach. In Andrzej Skowron, Jean-Paul A. Barth ?s, Lakhmi C. Jain, Ron Sun, Pierre Morizet-Mahoudeaux, Jiming Liu, and Ning Zhong, editors, *Proceedings of the 2005 IEEE/-WIC/ACM International Conference on Intelligent Agent Technology (IAT'05)*, pages 739–742, Compiègne, France, September 2005. IEEE Computer Society Press.

[101] Guillaume Muller, Laurent Vercouter, and Olivier Boissier. Towards a general definition of trust and its application to openness in mas. In R. Falcone, S. K. Barber, L. Korba, and M. Singh (eds), editors, *6th International Workshop on Trust, Privacy, Deception, and Fraud in Agent Societies*, Melbourne, Australia, 2003.

[102] Andrea Omicini. Soda : Societies and infrastructures in the analysis and design of agent-based systems. In Paolo Ciancarini and Michael Wooldridge, editors, *AOSE*, volume 1957 of *Lecture Notes in Computer Science*, pages 185–193. Springer, 2000.

[103] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Rbac for organisation and security in an agent coordination infrastructure. *Electron. Notes Theor. Comput. Sci.*, 128(5) :65–85, 2005.

[104] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. *Agens Faber* : Toward a theory of artefacts for MAS. *Electronic Notes in Theoretical Computer Sciences*, 150(3) :21–36, 29 May 2006.

[105] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3) :432–456, 2008.

[106] Joon S. Park, Ravi Sandhu, and Gail-Joon Ahn. Role-based access control on the web. *ACM Trans. Inf. Syst. Secur.*, 4(1) :37–71, 2001.

[107] G. Picard, J. F. Hubner, O. Boissier, and M.-P. Gleizes. Reorganisation and self-organisation in multi-agent systems. In *International Workshop on Organizational Modeling (OrgMod'09)*, 2009.

[108] G. Picard, J. F. Hubner, O. Boissier, and M.-P. Gleizes. Réorganisation et auto-organisation dans les systèmes multi-agents. In Zahia Guessoum and Salima Hassas, editors, *Journées Francophones sur les Systèmes Multi-Agents (JFSMA 2009)*, pages 89 – 98, Lyon, France, Octobre 2009. Cépaduès Editions.

[109] Gauthier Picard, Marie Pierre Gleizes, and Pierre Glize. Distributed frequency assignment using cooperative self-organization. In *SASO*, pages 183–192. IEEE Computer Society, 2007.

[110] Michele Piunti. *Designing and Programming Organizational Infrastructures for Agents situated in Artifact-based Environments*. PhD thesis, Universita di Bologna, 2007.

[111] Michele Piunti, Alessandro Ricci, Olivier Boissier, and Jomi Fred Hübner. Embodying organisations in multi-agent work environments. In *IAT*, pages 511–518. IEEE, 2009.

[112] David V. Pynadath and Milind Tambe. An automated teamwork infrastructure for heterogeneous software agents and humans. *Autonomous Agents and Multi-Agent Systems*, 7(1-2) :71–100, 2003.

[113] David V. Pynadath, Milind Tambe, Nicolas Chauvat, and Lawrence Cavedon. Toward team-oriented programming. In *ATAL '99 : 6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages*, pages 233–247, London, UK, 2000. Springer-Verlag.

[114] Anand S. Rao and Michael P. Georgeff. Bdi agents : From theory to practice. In *In proceedings of the first international conference on multi-agent systems (ICMAS-95)*, pages 312–319, 1995.

[115] Alessandro Ricci, Michele Piunti, L. Daghan Acay, Rafael H. Bordini, Jomi F. Hübner, and Mehdi Dastani. Integrating heterogeneous agent programming platforms within artifact-based environments. In *AAMAS '08 : Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 225–232, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[116] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. CArtAgO : A framework for prototyping artifact-based environments in MAS. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for MultiAgent Systems III*, volume 4389 of *LNAI*. Springer, 2006. 3rd International Workshop (E4MAS 2006).

[117] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. "Give agents their artifacts" : The A&A approach for engineering working environments in MAS. In Edmund Durfee, Makoto Yokoo, Michael Huhns, and Onn Shehory, editors, *6th International Joint Conference "Autonomous Agents & Multi-Agent Systems" (AAMAS 2007)*. IFAAMAS, 2007.

[118] Mansour Saber. *Un modèle de gestion distribuée de groupes ouverts et dynamiques d'agents mobiles*. PhD thesis, Université des Sciences et Techniques de Pau, Pau, France, Décembre 2007.

[119] Ravi Sandhu and Qamar Munawer. The arbac99 model for administration of roles. In *ACSAC '99 : Proceedings of the 15th Annual Computer Security Applications Conference*, page 229, Washington, DC, USA, 1999. IEEE Computer Society.

[120] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29 :38–47, 1996.

[121] Bruce Schneier. *Applied cryptography (2nd ed.) : protocols, algorithms, and source code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[122] W. Richard Scott. *Organizations : Rational, Natural, and Open Systems*. Prentice Hall, Englewood Cliffs, New Jersey, 1981.

[123] Yann Secq. *RIO : Rôles, Interactions et Organisations, une méthodologie pour les systèmes multi-agents ouverts*. PhD thesis, Université des Sciences et Technologies de Lille, Lille, France, Décembre 2003.

[124] Giovanna Di Marzo Serugendo, Marie Pierre Gleizes, and Anthony Karageorgos. Self-organisation and emergence in mas : An overview. *Informatica (Slovenia)*, 30(1) :45–54, 2006.

[125] Jaime Simão Sichman. Depint : Dependence-based coalition formation in an open multi-agent scenario. *J. Artificial Societies and Social Simulation*, 1(2), 1998.

[126] Gene Sparfford Simson Garfinkel. *Pratical Unix and Internet security*. O'Reilly & Associates, Inc.

[127] Katia P. Sycara. Multi-agent system. *Intelligent Agents*, 19(2), 1998.

[128] Milind Tambe and Weixiong Zhang. Towards flexible teamwork in persistent teams. *Journal of Autonomous Agents and Multi-Agent Systems*, 3 :159–183, 1998.

[129] Roy M. Turner. Context-mediated behavior for intelligent agents. *International Journal of Human-Computer Studies*, 48 :307–330, 1998.

[130] Rogier M. van Eijk, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Open multi-agent systems : Agent communication and integration. In Nicholas R. Jennings and Yves Lespérance, editors, *Agent Theories, Architectures, and Languages (ATAL)*, volume 1757 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 1999.

[131] Pierre-Yves Vandenbussche and Jean Charlet. Méta-modèle général de description de ressources terminologiques et ontologiques. In Fabien L. Gandon, editor, *Actes d'IC*, pages 193–204. PUG, 2009.

[132] Laurent Vercouter. *Conception et mise en oeuvre de systèmes multi-agents ouverts et distribués*. PhD thesis, École des mines de Saint-Etienne, 2000.

[133] Laurent Vercouter and Guillaume Muller. L.i.a.r. : Achieving social control in open and decentralized multiagent systems. *Applied Artificial Intelligence*, 24(8) :723–768, 2010.

[134] Bernard Walliser. *Systèmes et modèles : introduction critique à l'analyse de systèmes*. Seuil, Paris, 1977.

[135] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first-class abstraction in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, Online First, July 2006. Special Issue : Environment for Multi-Agent Systems.

[136] Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors. *Environments for Multi-Agent Systems II, Second International Workshop, E4MAS 2005, Utrecht, The Netherlands, July 25, 2005, Selected Revised and Invited Papers*, volume 3830 of *Lecture Notes in Computer Science*. Springer, 2006.

[137] Steven Willmott and Boi Faltings. The benefits of environment adaptive organisations for agent coordination and network routing problems. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS'2000)*, pages 333–340, Los Alamitos, CA, 2000. IEEE.

[138] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents : Theory and practice. *Knowledge Engineering Review*, 10 :115–152, 1995.

[139] Demazeau Yves. From interactions to collective behaviour in agent-based systems. In *Proceedings of the First European conference on cognitive science*, pages 117–132, Saint Malo, France, April 1995.

[140] Giorgos Zacharia and Pattie Maes. Trust management through reputation mechanisms. *Applied Artificial Intelligence*, 14(9) :881–907, 2000.

[141] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems : The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3) :317–370, 2003.

Liste des Publications

– Rosine Kitio, Olivier Boissier, Jomi Fred Hübner, and Alessandro Ricci. Organisational artifacts and agents for open multi-agent organisations : “giving the power back to the agents”. In Jaime Sichman, P. Noriega, J. Padget, and Sascha Ossowski, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems III*, volume 4870 of *LNCS*, pages 171–186. Springer, 2008. Revised Selected Papers.

– Rosine Kitio, Olivier Boissier, Jomi Fred Hübner, and Alessandro Ricci. Organisational artifacts and agents for open multi-agent organisations : “giving the power back to the agents”. In Mehdi Dastani and Rafael Bordinii, editors, *Fifth European Workshop on Multi-Agent Systems (EUMAS 2007)*. Paper initially published in COIN@AAMAS 2007.

– Olivier Boissier, Rosine Kitio, Jomi Fred Hübner, and Alessandro Ricci. Instrumentation d’organisations multi-agents avec des artefact organisationnels. In René Mandiau and Pierre Chevaillier, editors, *Actes des 16e Journées Francophones sur les Systèmes Multi-Agents (JFSMA’08)*, pages 107–116. Cépaduès, 2008.

– Jomi F. Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3) :369–400, 2010.

Annexe A

XML-Schema de MOISE

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:moise="http://moise.sourceforge.net/os"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified"
            targetNamespace="http://moise.sourceforge.net/os" >

<xsd:element name="organisational-specification" type="moise:osType"/>
<xsd:complexType name="osType">
<xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="0" name="properties" type="moise:propertiesType"/>
  <xsd:element maxOccurs="1" minOccurs="1" name="structural-specification" type="moise:ssType"/>
  <xsd:element maxOccurs="1" minOccurs="0" name="functional-specification" type="moise:fsType"/>
  <xsd:element maxOccurs="1" minOccurs="0" name="normative-specification" type="moise:nsType"/>
  <xsd:element maxOccurs="1" minOccurs="0" name="entryExit-specification" type="moise:eesType"/>
</xsd:sequence>
<xsd:attribute name="id" type="xsd:string"/>
<xsd:attribute name="os-version" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="ssType">
<xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="0" name="properties" type="moise:propertiesType"/>
  <xsd:element maxOccurs="1" minOccurs="0" name="role-definitions">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="role" type="moise:roleDefType"/>
   </xsd:sequence>
   </xsd:complexType>
   </xsd:element>

  <xsd:element maxOccurs="1" minOccurs="0" name="link-types">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="1" name="link-type">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:string" use="required"/>
    </xsd:complexType>
    </xsd:element>
```

```
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>

  <xsd:element maxOccurs="1" minOccurs="0" name="group-specification"
       type="moise:groupSpecificationType"/>

</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="fsType">
<xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="0" name="properties" type="moise:propertiesType"/>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="scheme">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="0" name="properties" type="moise:propertiesType"/>
    <xsd:element maxOccurs="1" minOccurs="1" name="goal" type="moise:goalDefType"/>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="mission">
    <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="0" name="properties" type="moise:propertiesType"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="adoptRequirementMission">
      <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="adoptReq"
                type="moise:requirementDefType"/>
      </xsd:sequence>
      </xsd:complexType>
      </xsd:element>

      <xsd:element maxOccurs="1" minOccurs="0" name="leaveRequirementMission">
      <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="leaveReq"
                type="moise:requirementDefType"/>
      </xsd:sequence>
      </xsd:complexType>
      </xsd:element>

      <xsd:element maxOccurs="unbounded" minOccurs="1" name="goal">
      <xsd:complexType>
        <xsd:attribute name="id" type="xsd:string" use="required"/>
      </xsd:complexType>
      </xsd:element>

      <xsd:element maxOccurs="unbounded" minOccurs="0" name="preferred">
      <xsd:complexType>
        <xsd:attribute name="mission" type="xsd:string" use="required"/>
      </xsd:complexType>
      </xsd:element>

    </xsd:sequence>
      <xsd:attribute name="id" type="xsd:string" use="required"/>
      <xsd:attribute name="min" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="max" type="xsd:nonNegativeInteger"/>
```

```
    </xsd:complexType>
    </xsd:element>
    </xsd:sequence>

  <xsd:attribute name="id" type="xsd:string"/>
  <xsd:attribute name="monitoring-scheme" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="nsType">
<xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="0" name="properties" type="moise:propertiesType"/>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="norm">
  <xsd:complexType>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
    <xsd:attribute name="condition" type="xsd:string" use="optional"/>
    <xsd:attribute name="role" type="xsd:string" use="required"/>
    <xsd:attribute name="type" type="xsd:string" use="required"/>
    <xsd:attribute name="mission" type="xsd:string" use="required"/>
    <xsd:attribute name="time-constraint" type="xsd:string" use="optional"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="propertiesType">
<xsd:sequence maxOccurs="unbounded" minOccurs="1">
  <xsd:element name="property">
  <xsd:complexType>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
    <xsd:attribute name="value" type="xsd:string" use="required"/>
  </xsd:complexType>
  </xsd:element>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="roleDefType">
<xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="0" name="properties" type="moise:propertiesType"/>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="extends">
  <xsd:complexType>
    <xsd:attribute name="role" type="xsd:string"/>
  </xsd:complexType>
  </xsd:element>

  <xsd:element maxOccurs="1" minOccurs="1" name="adoptRequirementRole">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="adoptReq"
        type="moise:requirementDefType"/>
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
```

```
  <xsd:element maxOccurs="1" minOccurs="0" name="leaveRequirementRole">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="leaveReq"
    type="moise:requirementDefType"/>
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
</xsd:sequence>
  <xsd:attribute name="id" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:simpleType name="cardinalityObjectType">
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="role"/>
  <xsd:enumeration value="group"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="groupSpecificationType">
 <xsd:attribute name="id" type="xsd:string" use="required"/>
 <xsd:attribute name="min" type="xsd:nonNegativeInteger" use="optional"/>
 <xsd:attribute name="max" type="xsd:nonNegativeInteger" use="optional"/>
 <xsd:attribute name="monitoring-scheme" type="xsd:string"/>

 <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="0" name="roles">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="1" name="role">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:string"/>
      <xsd:attribute name="min" type="xsd:nonNegativeInteger" use="optional"/>
      <xsd:attribute name="max" type="xsd:nonNegativeInteger" use="optional"/>
    </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>

  <xsd:element maxOccurs="1" minOccurs="0" name="links">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="1" name="link">
    <xsd:complexType>
      <xsd:attribute name="from" type="xsd:string"/>
      <xsd:attribute name="to" type="xsd:string"/>
      <xsd:attribute name="type" type="xsd:string"/>
      <xsd:attribute default="intra-group" name="scope" type="moise:scopeType"/>
      <xsd:attribute default="false" name="extends-sub-groups" type="xsd:boolean"/>
      <xsd:attribute default="false" name="bi-dir" type="xsd:boolean"/>
    </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
```

```
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="sub-groups">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="include-group-specification">
    <xsd:complexType>
      <xsd:attribute name="uri" type="xsd:string"/>
    </xsd:complexType>
    </xsd:element>

    <xsd:element maxOccurs="unbounded" minOccurs="0" name="group-specification"
        type="moise:groupSpecificationType"/>
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>

  <xsd:element maxOccurs="1" minOccurs="0" name="formation-constraints">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="cardinality">
    <xsd:complexType>
      <xsd:attribute name="min" type="xsd:nonNegativeInteger" use="optional"/>
      <xsd:attribute name="max" type="xsd:nonNegativeInteger" use="optional"/>
      <xsd:attribute name="object" type="moise:cardinalityObjectType" use="required"/>
      <xsd:attribute name="id" type="xsd:string" use="required"/>
    </xsd:complexType>
    </xsd:element>

    <xsd:element maxOccurs="unbounded" minOccurs="0" name="compatibility">
    <xsd:complexType>
      <xsd:attribute name="from" type="xsd:string" use="required"/>
      <xsd:attribute name="to" type="xsd:string" use="required"/
      <xsd:attribute default="intra-group" name="scope" type="moise:scopeType"/>
      <xsd:attribute default="false" name="extends-sub-groups" type="xsd:boolean"/>
      <xsd:attribute default="false" name="bi-dir" type="xsd:boolean"/>
      <xsd:attribute name="type" type="xsd:string"/>
    </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>

</xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="scopeType">
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="intra-group"/>
  <xsd:enumeration value="inter-group"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="planType">
 <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="0" name="properties" type="moise:propertiesType"/>
  <xsd:element maxOccurs="unbounded" minOccurs="1" name="goal" type="moise:goalDefType"/>
```

```
 </xsd:sequence>

 <xsd:attribute name="operator" type="moise:planOperatorType" use="required"/>
 <xsd:attribute name="success-rate" type="xsd:double" use="optional"/>
</xsd:complexType>

<xsd:complexType name="goalDefType">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="argument" type="moise:argumentType"/>
  <xsd:element maxOccurs="1" minOccurs="0" name="plan" type="moise:planType"/>
  <xsd:element maxOccurs="1" minOccurs="0" name="adoptRequirementGoal">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="adoptReq"
        type="moise:requirementDefType"/>
   </xsd:sequence>
   </xsd:complexType>
   </xsd:element>

  <xsd:element maxOccurs="1" minOccurs="0" name="leaveRequirementGoal">
  <xsd:complexType>
  <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="leaveReq"
        type="moise:requirementDefType"/>
   </xsd:sequence>
   </xsd:complexType>
   </xsd:element>
 </xsd:sequence>

 <xsd:attribute name="id" type="xsd:string" use="required"/>
 <xsd:attribute name="min" type="xsd:nonNegativeInteger" use="optional"/>
 <xsd:attribute name="ds" type="xsd:string" use="optional"/>
 <xsd:attribute name="type" type="moise:goalType"/>
 <xsd:attribute name="ttf" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:complexType name="argumentType">
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:attribute name="value" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:simpleType name="planOperatorType">
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="sequence"/>
  <xsd:enumeration value="choice"/>
  <xsd:enumeration value="parallel"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="goalType">
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="achievement"/>
  <xsd:enumeration value="maintenance"/>
</xsd:restriction>
</xsd:simpleType>
```

```
<xsd:complexType name="requirementDefType">
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:attribute name="type" type="moise:requirementType" use="required"/>
  <xsd:attribute name="expression" type="xsd:string" use="required"/>
  <xsd:attribute name="property" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:simpleType name="requirementType">
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="AdoptReq"/>
  <xsd:enumeration value="LeaveReq"/>
  <xsd:enumeration value="EntryReq"/>
  <xsd:enumeration value="ExitReq"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="eesType">
<xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="0" name="properties" type="moise:propertiesType"/>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="portal">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="0" name="properties" type="moise:propertiesType"/>
    <xsd:element maxOccurs="1" minOccurs="0" name="entry-requirements">
    <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="entryReq"
        type="moise:requirementDefType"/>
    </xsd:sequence>
    </xsd:complexType>
    </xsd:element>

    <xsd:element maxOccurs="1" minOccurs="0" name="exit-requirements">
    <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="exitReq"
        type="moise:requirementDefType"/>
    </xsd:sequence>
    </xsd:complexType>
    </xsd:element>

    <xsd:element maxOccurs="unbounded" minOccurs="0" name="gates">
    <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="gate">
      <xsd:complexType>
        <xsd:attribute name="id" type="xsd:string" use="required"/>
        <xsd:attribute name="target" type="xsd:string"/>
        <xsd:attribute name="groupId" type="xsd:string"/>
        <xsd:attribute name="schemeId" type="xsd:string"/>
        <xsd:attribute name="deadline" type="xsd:string"/>
      </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    </xsd:complexType>
    </xsd:element>
```

```
    <xsd:element maxOccurs="unbounded" minOccurs="1" name="associated-groups">
    <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="1" name="associated-group">
      <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" name="gate">
        <xsd:complexType>
          <xsd:attribute name="id" type="xsd:string" use="required"/>
        </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
        <xsd:attribute name="id" type="xsd:string" use="required"/>
        <xsd:attribute name="group" type="xsd:string"/>
        <xsd:attribute name="portalType" type="xsd:string"/>
      </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
   <xsd:attribute name="id" type="xsd:string" use="required"/>
  </xsd:complexType>
  </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="eePortalType">
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="Entry"/>
  <xsd:enumeration value="Exit"/>
  <xsd:enumeration value="EntryExit"/>
</xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

Annexe B

Spécification organisationnelle XML de l'exemple Write-Paper

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="/Users/kitio/Ora4mas-Workspace/ORA4MAS.V.1.0/xml/os.xsl" type="text/xsl" ?>

<organisational-specification
    id="wp"
    os-version="0.8.1"
    xmlns='http://moise.sourceforge.net/os'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://moise.sourceforge.net/os/xml/os.xsd' >
 <structural-specification>
  <role-definitions>
    <role id="author">
      <adoptRequirementRole/>
      <leaveRequirementRole/>
    </role>

    <role id="writer">
    <extends role="author"/>
      <adoptRequirementRole>
        <adoptReq id="ad_req_01" type="AdoptReq" expression="DomainSkill==MAS_organisation"
                  property="mandatory"/>

        <adoptReq id="ad_req_02" type="AdoptReq" expression="LanguageSkill==English"
                  property="mandatory"/>
      </adoptRequirementRole>
      <leaveRequirementRole/>
    </role>

    <role id="editor">
    <extends role="author"/>
      <adoptRequirementRole>
        <adoptReq id="ad_req_01" type="AdoptReq" expression="DomainSkill==MAS_organisation"
                  property="mandatory"/>
```

```
        <adoptReq id="ad_req_02" type="AdoptReq" expression="LanguageSkill==English"
                  property="mandatory"/>
      </adoptRequirementRole>
      <leaveRequirementRole/>
    </role>
  </role-definitions>

  <group-specification id="wpgroup" monitoring-scheme="monitoringSch">
    <roles>
      <role id="writer" min="1" max="5" />
      <role id="editor" min="1" max="1" />
    </roles>

    <links>
      <link from="writer" to="editor" type="acquaintance"
            scope="intra-group" extends-sub-groups="true" bi-dir="false"/>
      <link from="editor" to="writer" type="authority"
            scope="intra-group" extends-sub-groups="true" bi-dir="false"/>
      <link from="author" to="author" type="communication"
            scope="intra-group" extends-sub-groups="true" bi-dir="false"/>
    </links>

    <formation-constraints>
      <compatibility from="editor" to="writer" type="compatibility"
            scope="intra-group" extends-sub-groups="false" bi-dir="true"/>
    </formation-constraints>
  </group-specification>
</structural-specification>

<functional-specification>
  <scheme id="writePaperSch" monitoring-scheme="monitoringSch">
    <goal id="wp" ttf="5 seconds">
    <plan operator="sequence" >
      <goal id="fdv" ds="First Draft Version">
      <plan operator="sequence">
        <goal id="wtitle"  ttf="1 day" ds="Write a title">
          <adoptRequirementGoal/>
          <leaveRequirementGoal/>
        </goal>

        <goal id="wabs" ttf="1 day" ds="Write an abstract">
          <adoptRequirementGoal/>
          <leaveRequirementGoal/>
        </goal>

        <goal id="wsectitles" ttf="1 day" ds="Write the sections' title">
          <adoptRequirementGoal/>
          <leaveRequirementGoal/>
        </goal>
      </plan>
      </goal>

      <goal id="sv" ds="Submission Version">
      <plan operator="sequence">
        <goal id="wsecs"  ttf="7 days" ds="Write sections">
          <adoptRequirementGoal/>
```

```
        <leaveRequirementGoal/>
       </goal>

       <goal id="finish" ds="Finish paper">
       <plan operator="parallel">
         <goal id="wcon"  ttf="1 day"  ds="Write a conclusion">
           <adoptRequirementGoal/>
           <leaveRequirementGoal/>
         </goal>

         <goal id="wrefs" ttf="1 hour" ds="Complete references and link them to text">
           <adoptRequirementGoal/>
           <leaveRequirementGoal/>
         </goal>
       </plan>
       </goal>
    </plan>
    </goal>
 </plan>
 </goal>

 <mission id="mCol" min="1" max="5">
   <adoptRequirementMission/>
   <leaveRequirementMission>
     <leaveReq id="le_req_11" type="LeaveReq" expression="hasFinishedAllGoal == _"
               property="mandatory"/>
    </leaveRequirementMission>

   <goal id="wsecs"/>
 </mission>

 <mission id="mMan" min="1" max="1">
   <adoptRequirementMission>
     <adoptReq id="ad_req_03" type="AdoptReq" expression="hasCapability == Management"
               property="optional"/>
    </adoptRequirementMission>

   <leaveRequirementMission>
     <leaveReq id="le_req_11" type="LeaveReq" expression="hasFinishedAllGoal == _"
               property="mandatory"/>
    </leaveRequirementMission>

   <goal id="wabs"/>
   <goal id="wtitle"/>
   <goal id="wcon"/>
   <goal id="wsectitles"/>
 </mission>

<mission id="mBib" min="1" max="1">
  <adoptRequirementMission/>
  <leaveRequirementMission>
    <leaveReq id="le_req_11" type="LeaveReq" expression="hasFinishedAllGoal == _"
              property="mandatory"/>
   </leaveRequirementMission>

   <goal id="wrefs"/>
```

```
      <preferred mission="mCol"/>
      <preferred mission="mMan"/>
    </mission>
  </scheme>

  <scheme id="monitoringSch">
    <goal id="monitor">
    <plan operator="choice">
      <goal id="sanction" ds="Sanction the agent that is not doing its job!"/>
      <goal id="reward"   ds="Reward some agent for doing a good job!"/>
    </plan>
    </goal>
    <mission id="mSan" min="1" max="1" >
      <goal id="sanction"/>
    </mission>
    <mission id="mRew" min="1" max="1" >
      <goal id="reward"/>
    </mission>
  </scheme>
</functional-specification>

 <normative-specification>
   <norm id="n1" type="permission" role="editor" mission="mMan" />
   <norm id="n2" type="obligation" role="writer" mission="mCol" time-constraint="2 day"/>
   <norm id="n3" type="obligation" role="writer" mission="mBib" time-constraint="1 day"/>
   <norm id="n4" type="obligation" condition="fulfilled(n2)"
                 role="editor" mission="mRew" time-constraint="3 hours"/>
   <norm id="n5" type="obligation" condition="unfulfilled(n3)"
                 role="editor" mission="mSan" time-constraint="3 hours"/>
 </normative-specification>

 <entryExit-specification >
   <portal id="wpPor">
     <entry-requirements>
       <entryReq id="en_req1" type="EntryReq" expression="HasSkill==MAS" property="mandatory"/>
     </entry-requirements>
     <exit-requirements/>

     <gates>
       <gate id="ga1" target="(writer, _ , _ )" groupId="wpgroup"
                 schemeId="writePaperSch" deadline="3weeks"/>
       <gate id="ga2" target="(editor, mMan, _ )" groupId="wpgroup"
                 schemeId="writePaperSch" deadline="1weeks"/>
     </gates>

     <associated-groups>
       <associated-group id="assGr1" group="wpgroup" portalType="entryExit">
         <gate id="ga1"/>
         <gate id="ga2"/>
       </associated-group>
     </associated-groups>
   </portal>
 </entryExit-specification>

</organisational-specification>
```

Annexe C

Codes de quelques agents Jason développés pour l'exemple Write-Paper

C.1 Code de l'OrgAgent

```
// ORA4MAS Experimentation : Write Paper example
// OrgAgent code
/* Initial goals */
!create_org.

/* Plans */
// plan to achieve the goal 'create an organisation entity'
+!create_org : true
      <- .print("Creation of ORA4MAS organisational artifacts!");
         // enter in Workspace named "ORA4MAS-WS1" located at localhost:3047
      cartago.joinWorkspace("ORA4MAS-WS1","localhost:3047");

  // create and configure OrgBoard
      cartago.makeArtifact("writePaperOrgBoard1", "ora4masMoise.OrgBoard.OrgBoard",
       ["./XmlFilesOrgSpecif/write-paperOS.xml"], OId);
   cartago.use(OId, configure, s1);
  //perceive the event generated by the operation configure
  cartago.sense(s1, initOrgBoard_ev(OrgB,O1,Res1));
  if (Res1 == "succeeded") {
   +orgBoardId(OId);
   .print("Creation of OrgBoard: " , OId, " succeeded")
  };

  // create and configure GroupBoard
  cartago.makeArtifact("writePaperGrpBoard1", "ora4masMoise.GroupBoard.GroupBoard",  GId);
  cartago.use(GId, configure(OId, "wpgroup", ""), s2);
  cartago.sense(s2, initGroupBoard_ev(GrpB,O2,Res2));
  if (Res2 == "succeeded") {
```

```
 +grpBoardId(GId);
 .print(" Creation of GroupBoard: " , GId, " succeeded")

};

// create and configure SchemeBoard
cartago.makeArtifact("writePaperSchBoard1", "ora4masMoise.SchemeBoard.SchemeBoard", SId);
cartago.use(SId, configure(OId, "writePaperSch", GId), s3);
cartago.sense(s3, initSchemeBoard_ev(SchB,O3,Res3));
if (Res3 == "succeeded") {
 +schBoardId(SId);
 .print("Creation of SchemeBoard: " , SId, " succeeded")
 //out.print(" ")
};

// create and configure SchemeBoard
cartago.makeArtifact("writePaperMonitoringSchBoard1", "ora4masMoise.SchemeBoard.SchemeBoard", SId2);
cartago.use(SId2, configure(OId, "monitoringSch", GId), s3);
cartago.sense(s3, initSchemeBoard_ev(SchB,O4,Res3));
if (Res3 == "succeeded") {
 +schBoardId(SId2);
 .print("Creation of the monitoring SchemeBoard: ", SId2, " succeeded")
 //out.print(" ")
};


// create and configure PortalBoard
cartago.makeArtifact("writePaperPorBoard1", "ora4masMoise.PortalBoard.PortalBoard", PId);
cartago.use(PId, configure(OId, "wpPor"),s5);
cartago.sense(s5, initPortalBoard_ev(PorB,O5,Res5));
if (Res5 == "succeeded") {
 +porBoardId(PId);
 .print("Creation of PortalBoard: " , PId, " succeeded");
 .print(" ")
};

//set portal of GroupBoard writePaperGrpBoard1
cartago.use(GId, addPortalBoard(PId, entryExit), s2);
.print("Set PortalBoard: " , PId, " as manager of Entry-Exit in ", GId);
.print(" ");

//create the call for agent candidature for
//the gates ga1 and ga2 of writePaperPorBoard1
cartago.use(PId, createCFAC(cfac1, ga1, GId, SId),s5);
cartago.use(PId, createCFAC(cfac2, ga2, GId, SId),s);
.print("Succeeded creation of CFAC1 and CFAC2 on PortalBoard: ", PId);
.print(" ");

//focus all the created organisationnal artifacts
cartago.focus(OId);
cartago.focus(GId);
cartago.focus(SId);
cartago.focus(SId2);
cartago.focus(PId);

//Send to agent bob and alice message to achieve the goal enterOrg
```

```
  .send([jeanne], achieve, enterOrg);
  .send([bob], achieve, enterOrg);
  .send([alice], achieve, enterOrg);
  .print("Send to jeanne, bob and alice message enterOrg");
  .print(" ").

+submitCandidature_ev(P1,"alice","succeeded")
: porBoardId(PId)
<- .print("Succeeded submit candidature ev ", P1, " ",alice);
cartago.use(PId, createContract(cfac1, alice),s1).

+submitCandidature_ev(P1,"jeanne","succeeded")
: porBoardId(PId)
<- .print("Succeeded submit candidature ev ", P1, " ",jeanne);
cartago.use(PId, createContract(cfac1, jeanne),s1).

+submitCandidature_ev(P1,"bob","succeeded")
: porBoardId(PId)
<- .print("Succeeded submit candidature ev ", P1, " ",bob);
cartago.use(PId, createContract(cfac2, bob),s1).

+registerContractInGroupBoardLinkOp_ev(G1, Owner,"succeeded")
<- .print("succeeded recording Contract in ", G1, " ", Owner);
   .send([Owner], achieve, contractCreated).

+commitMission_ev(S,O, M,"succeeded", Ag, "normViolation")
 : schBoardId(SId)
    <- .print(Ag," violate a norm by commit to Mission ", M, " on ", S);
       .send([Ag], achieve, leaveMissionNormViolation);
   .print(" ").

+commitMission_ev(S,O,M,"succeeded",Ag) : schBoardId(SId)
    <- .print(Ag," succeeded commitMission ", M, " on ", S);
   .print(" ").

+leaveMission_ev(S,O,"mMan","succeeded", "jeanne")
 : schBoardId(SId)
    <- .print(jeanne," leave the Mission ", mMan, " on ", S);
     .send([bob], achieve, commitMissionNow);
   .print(" ").

+leaveMission_ev(S,O,"mMan","succeeded", "alice")
 : schBoardId(SId)
    <- .print(alice," leave the Mission ", "mMan", " on ", S);
     .send([bob], achieve, commitMissionNow);
   .print(" ").

+newPossibleGoal_ev(S,Op,Goal)
  <- .print("It is now possible to achieve the goal : ", Goal, " on ", S).

+setGoalAchieved_ev(S,O,G,"succeeded",Ag)
    <- .print(Ag, "succeeded the operation setGoalAchieved on OrgArt ", S,
     " for the goal ", G);
   .print(" ").

+setGoalAchieved_ev(S,O,G,"failed",Ag,R)
```

```
    <- .print(Ag, " failed the operation setGoalAchieved on OrgArt ", S,
     " for the goal ", G, " with the reason ", R);
   .print(" ").

+commitMission_ev(S,O,M,"failed", Ag, R)
 <- .print(Ag," failed commitMission ", M, " on ", S, " with the reason ", R);
.print(" ").

+registerContractInGroupBoardLinkOp_ev(G1,OpUser,"failed")
<- .print("Failed Contract recording in ", G1, " ", OpUser).
```

C.2 Code d'un DomainAgent : editor

```
// Agent02 in project WRITE_PAPER.mas2j
// It will candidate to cfac2
// If it is admited, it will commit to mission mMan

/* Plans */

+!enterOrg
      <-  cartago.joinWorkspace("ORA4MAS-WS1","localhost:3047");
  .print("enter org: ");
  cartago.lookupArtifact("writePaperGrpBoard1", GId);
      +grpBoardId(GId);
  cartago.focus(GId);

  cartago.lookupArtifact("writePaperSchBoard1", SId);
  +schBoardId(SId);
  cartago.focus(SId);

  cartago.lookupArtifact("writePaperPorBoard1", PId);
  +porBoardId(PId);
  cartago.focus(PId);
  cartago.use(PId, submitCandidature(cfac2, "{(en_req1: hasSkill ; MAS)}
  {(ad_req01: DomainSkill ; MAS_environment, MAS_organisation),
  (ad_req02: LanguageSkill ; English), (ad_req03 : hasCapability ; Management)}"), s5);
  .print("Candidate to CFAC2 ").

+registerContractInGroupBoardLinkOp_ev(G1,OpUser,"succeeded")
<- .print("Success recording Contract in ", G1, " ", OpUser);
.send([OpUser], achieve, contractCreated);
.print(" send msg ").

+registerContractInGroupBoardLinkOp_ev(G1,OpUser,"failed")
<- .print("Failed recording Contract in ", G1, " ", OpUser).

//plan to commit mission in case of contract creation
+!contractCreated  : schBoardId(SId)
      <- .print("Receive msg contractCreated");
         cartago.use(SId, commitMission(mMan));
  .print("Commit to mission mManager");
  .print(" ").
```

```
//plan to commit mission in case of mission free
+!commitMissionNow : schBoardId(SId)
      <- .print("Receive msg commitMissionNow");
         cartago.use(SId, commitMission(mMan));
  .print("Commit to mission mManager");
  .print(" ").

+newPossibleGoal_ev(S1,Op1,"wtitle")
: schBoardId(SId)
    <- cartago.use(SId, setGoalAchieved(wtitle)).

+newPossibleGoal_ev(S1,Op1,"wabs")
: schBoardId(SId)
    <-  .print("Goal 'wtitle' achieved");
     cartago.use(SId, setGoalAchieved(wabs)).

+newPossibleGoal_ev(S1,Op1,"wsectitles")
: schBoardId(SId)
    <-  .print("Goal 'wabs' achieved");
     cartago.use(SId, setGoalAchieved(wsectitles), s1).

+newPossibleGoal_ev(S1,Op1,"wcon")
: schBoardId(SId)
    <- cartago.use(SId, setGoalAchieved(wcon), s1).
```

C.3 Code d'un DomainAgent : writer (1)

```
// Agent04 : will play "writer" role in project WRITE_PAPER.mas2j
// It will candidate to cfac1
// If it is admitted, it will commit to mission mCol and mMan

/* Plans */

+!enterOrg
   <-  cartago.joinWorkspace("ORA4MAS-WS1","localhost:3047");
       .print("Enter Org: ");
  cartago.lookupArtifact("writePaperGrpBoard1", GId);
      +grpBoardId(GId);
  cartago.focus(GId);

  cartago.lookupArtifact("writePaperSchBoard1", SId);
  +schBoardId(SId);
  cartago.focus(SId);

  cartago.lookupArtifact("writePaperPorBoard1", PId);
  +porBoardId(PId);

  cartago.use(PId, submitCandidature(cfac1, "{(en_req1: hasSkill ; MAS)}
  {(ad_req01: DomainSkill ; MAS_environment, MAS_organisation),
  (ad_req02: LanguageSkill ; English)}"), s5);

  .print("Submit a candidature for CFAC1 ").
```

```
+newPossibleGoal_ev(S1,Op1,"wsecs") : schBoardId(SId)
    <- cartago.use(SId, setGoalAchieved(wsecs), s1);
     cartago.sense(s1, setGoalAchieved_ev(S2,Op2,Goal, Res, Ag));

+!contractCreated  : schBoardId(SId)
      <- .print("Received msg contractCreated");
         cartago.use(SId, commitMission(mCol));
  .print("Committed to mission mCol");
  .print(" ");
  cartago.use(SId, commitMission(mMan));
  .print("Committed to mission mMan");
  .print(" ").

+!leaveMissionNormViolation  : schBoardId(SId)
      <- .print("Received msg leaveMissionManager");
         cartago.use(SId, leaveMission(mMan));
  .print("Leave mission mMan");
  .print(" ").
```

C.4 Code d'un DomainAgent : writer (2)

```
// Agent03 : will play "writer" role in project WRITE_PAPER.mas2j
// It will candidate to cfac1
// If it is admitted, it will commit to mission mClo and mBib

/* Plans */

+!enterOrg
   <-  cartago.joinWorkspace("ORA4MAS-WS1","localhost:3047");
       .print("Enter Org: ");
  cartago.lookupArtifact("writePaperGrpBoard1", GId);
      +grpBoardId(GId);
  cartago.focus(GId);

  cartago.lookupArtifact("writePaperSchBoard1", SId);
  +schBoardId(SId);
  cartago.focus(SId);

  cartago.lookupArtifact("writePaperPorBoard1", PId);
  +porBoardId(PId);

  cartago.use(PId, submitCandidature(cfac1, "{(en_req1: hasSkill ; MAS)}
  {(ad_req01: DomainSkill ; MAS_environment, MAS_organisation),
  (ad_req02: LanguageSkill ; English)}"), s5);

  .print("Submit a candidature for CFAC1 ").

+!contractCreated  : schBoardId(SId)
      <- .print("Received msg ");
          cartago.use(SId, commitMission(mCol));
         .print(" ");
  .print("Committed to mission mCol");
   cartago.use(SId, commitMission(mBib));
```

```
        .print(" ");
  .print("Committed to mission mBib");
  .print(" ").

+newPossibleGoal_ev(S1,Op1,"wsecs") : schBoardId(SId)
    <- cartago.use(SId, setGoalAchieved(wsecs), s1).

+newPossibleGoal_ev(S1,Op1,"wrefs") : schBoardId(SId)
    <- cartago.use(SId, setGoalAchieved(wrefs), s1).

+setGoalAchieved_ev(S,O,G,"succeeded",Ag)
    <- .print(Ag, " succeeded achieved goal ", G, " on OrgArt ", S);
   .print(" ").

+setGoalAchieved_ev(S,O,G,"failed",Ag,R)
    <- .print(Ag, " failed achieved goal ", G, " on OrgArt ", S);
   .print(" ").

+newPossibleGoal_ev(S,Op,Goal)
  <- .print("It is now possible to achieve the goal : ", Goal, " on ", S).

+!leaveMissionNormViolation  : schBoardId(SId)
      <- .print("Received msg leaveMissionManager");
         cartago.use(SId, leaveMission(mMan));
  .print("Leave mission mMan");
  .print(" ").
```

Annexe D

Spécification organisationnelle XML du cas d'étude Build-House

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="/Users/kitio/Ora4mas-Workspace/ORA4MAS.V.1.0/xml/os.xsl" type="text/xsl" ?>
<organisational-specification
    id="wp"
    os-version="0.8.1"
    xmlns='http://moise.sourceforge.net/os'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://moise.sourceforge.net/os/xml/os.xsd' >
 <structural-specification>
  <role-definitions>
    <role id="house_owner">
      <adoptRequirementRole/>
      <leaveRequirementRole/>
    </role>

    <role id="building_company">
      <adoptRequirementRole/>
      <leaveRequirementRole/>
    </role>

    <role id="site_prep_contractor">
    <extends role="building_company"/>
      <adoptRequirementRole>
        <adoptReq id="ar1r2" type="AdoptReq" property="mand" expression="3&lt;=team_numb&lt;=5"/>
      </adoptRequirementRole>
      <leaveRequirementRole/>
    </role>

   <role id="bricklayer">
   <extends role="building_company"/>
     <adoptRequirementRole>
       <adoptReq id="ar1r3" type="AdoptReq" property="mand" expression="5&lt;=team_numb&lt;=7"/>
     </adoptRequirementRole>
     <leaveRequirementRole/>
   </role>
```

```
<role id="roofer" >
<extends role="building_company"/>
  <adoptRequirementRole>
    <adoptReq id="ar1r4" type="AdoptReq" property="mand" expression="4&lt;=team_numb&lt;=10"/>
  </adoptRequirementRole>
  <leaveRequirementRole/>
</role>

<role id="plasterer">
<extends role="building_company"/>
  <adoptRequirementRole>
    <adoptReq id="ar1r5" type="AdoptReq" property="mand" expression="2&lt;team_numb&lt;=5"/>
  </adoptRequirementRole>
  <leaveRequirementRole/>
</role>

<role id="plumber">
<extends role="building_company"/>
  <adoptRequirementRole>
    <adoptReq id="ar1r6" type="AdoptReq" property="mand" expression="2&lt;=team_numb&lt;=3"/>
  </adoptRequirementRole>
  <leaveRequirementRole/>
</role>

<role id="electrician">
<extends role="building_company"/>
  <adoptRequirementRole>
    <adoptReq id="ar1r7" type="AdoptReq" property="mand" expression="1&lt;=team_numb&lt;=6"/>
  </adoptRequirementRole>
  <leaveRequirementRole/>
</role
<role id="tiles_layer">
<extends role="building_company"/>
  <adoptRequirementRole>
    <adoptReq id="ar1r9" type="AdoptReq" property="mand" expression="4&lt;=team_numb&lt;=7"/>
  </adoptRequirementRole>
  <leaveRequirementRole/>
</role>

<role id="joiner">
<extends role="building_company"/>
  <adoptRequirementRole>
    <adoptReq id="ar1r8" type="AdoptReq" property="mand" expression="3&lt;=team_numb&lt;=5"/>
  </adoptRequirementRole>
  <leaveRequirementRole/>
</role>

<role id="frontage_maker">
<extends role="building_company"/>
  <adoptRequirementRole>
    <adoptReq id="ar1r10" type="AdoptReq" property="mand" expression="4&lt;=team_numb&lt;=10"/>
  </adoptRequirementRole>
  <leaveRequirementRole/>
</role>
</role-definitions>
```

```
<group-specification id="building_grp">
 <roles>
   <role id="house_owner" min="1" max="1"/>
 </roles>

<links>
  <link from="house_owner" to="building_company" type="authority" scope="intra-group"
      extends-sub-groups="false" bi-dir="false"/>
  <link from="building_company" to="house_owner" type="communication" scope="intra-group"
      extends-sub-groups="false" bi-dir="false"/>
  <link from="building_company" to="building_company" type="communication" scope="inter-group"
      extends-sub-groups="true" bi-dir="true"/>
  <link from="building_company" to="building_company" type="communication" scope="intra-group"
      extends-sub-groups="true" bi-dir="true"/>
</links>

<sub-groups>
  <group-specification id="bigworks_grp" min="1" max="1">
    <roles>
      <role id="site_prep_contractor" min="1" max="2"/>
      <role id="bricklayer"           min="1" max="1"/>
      <role id="roofer"               min="1" max="2"/>
    </roles>

    <formation-constraints>
      <compatibility from="site_prep_contractor" to="bricklayer" type="compatibility"
         scope="intra-group" extends-sub-groups="false" bi-dir="false"/>
    </formation-constraints>
 </group-specification>

 <group-specification id="fitting_grp" min="1" max="1">
   <roles>
     <role id="plasterer"            min="1" max="3"/>
     <role id="plumber"              min="1" max="1"/>
     <role id="electrician"          min="1" max="1"/>
   </roles>

   <formation-constraints>
     <compatibility from="plasterer" to="tiles_layer" type="compatibility"
       scope="inter-group" extends-sub-groups="false" bi-dir="false"/>
   </formation-constraints>
 </group-specification>

 <group-specification id="finition_grp" min="1" max="1">
   <roles>
     <role id="joiner"               min="1" max="2"/>
     <role id="tiles_layer"          min="1" max="1"/>
     <role id="frontage_maker"       min="1" max="1"/>
   </roles>
 </group-specification>
</sub-groups>

 </group-specification>
</structural-specification>
```

```
<functional-specification>
  <scheme id="build_house_sch">
   <goal id="g0" ds="house_built">
   <plan operator="sequence">
     <goal id="g01" ds="first_step">
     <plan operator="sequence">
       <goal id="g1a" ds="excavation" ttf="1 week">
         <adoptRequirementGoal>
           <adoptReq id="ar1g1a" type="AdoptReq" property="mand" expression="goal_cost == _"/>
         </adoptRequirementGoal>
         <leaveRequirementGoal/>
       </goal>

       <goal id="g11">
       <plan operator="parallel">
         <goal id="g2a" ds="fondation_built" ttf="4 week">
           <adoptRequirementGoal>
             <adoptReq id="ar1g1a" type="AdoptReq" property="mand" expression="goal_cost == _"/>
           </adoptRequirementGoal>
           <leaveRequirementGoal/>
         </goal>

         <goal id="g1b" ds="roal_sewer_system" ttf="3 week">
           <adoptRequirementGoal>
             <adoptReq id="ar1g1b" type="AdoptReq" property="mand" expression="goal_cost == _"/>
           </adoptRequirementGoal>
           <leaveRequirementGoal/>
         </goal>
       </plan>
       </goal>

       <goal id="g12">
       <plan operator="parallel">
         <goal id="g2b" ds="walls_built" ttf="6 week">
           <adoptRequirementGoal>
             <adoptReq id="ar1g2b" type="AdoptReq" property="mand" expression="goal_cost == _"/>
           </adoptRequirementGoal>
           <leaveRequirementGoal/>
        </goal>
        <goal id="g5a" ds="plumber_incorporation" ttf="2 week"/>
        <goal id="g6a" ds="electrician_incorporation" ttf="2 week"/>
       </plan>
       </goal>

      <goal id="g02">
      <plan operator="parallel">
        <goal id="g13">
        <plan operator="sequence">
          <goal id="g3a" ds="roof-structure_built" ttf="2 week">
            <adoptRequirementGoal>
              <adoptReq id="ar1g3a" type="AdoptReq" property="mand" expression="goal_cost == _"/>
              <adoptReq id="ar2g3a" type="AdoptReq" property="mand"
                  expression="has_standard == NF P 21-203-1"/>
           </adoptRequirementGoal>
           <leaveRequirementGoal/>
         </goal>
```

```
        <goal id="g3b" ds="roof_built" ttf="2 week">
          <adoptRequirementGoal>
            <adoptReq id="ar1g3b" type="AdoptReq" property="mand" expression="goal_cost == _"/>
            <adoptReq id="ar2g3b" type="AdoptReq" property="mand"
                expression="has_standard == NF P 31-301"/>
          </adoptRequirementGoal>
          <leaveRequirementGoal/>
        </goal>
      </plan>
      </goal>
      <goal id="g4a" ds="steelframe_built" ttf="1 week" />
    </plan>
    </goal>
  </plan>
  </goal>

  <goal id="g03" ds="second_step">
  <plan operator="sequence">
    <goal id="g14">
    <plan operator="parallel">
      <goal id="g4b" ds="isolation_made" ttf="6 week"/>
      <goal id="g5b" ds="plumber_installed" ttf="4 week"/>
      <goal id="g6b" ds="electrician_installed" ttf="4 week"/>
    </plan>
    </goal>

    <goal id="g15">
    <plan operator="parallel">
      <goal id="g7a" ds="door_fitted" ttf="1 week"/>
      <goal id="g7b" ds="windows_fitted" ttf="1 week"/>
      <goal id="g9a" ds="frontage_fitted" ttf="2 week">
        <adoptRequirementGoal>
          <adoptReq id="ar1g9a" type="AdoptReq" property="mand" expression="goal_cost == _"/>
          <adoptReq id="ar2g9a" type="AdoptReq" property="mand"
              expression="has_standard == DTU 59.1"/>
        </adoptRequirementGoal>
        <leaveRequirementGoal/>
      </goal>
    </plan>
    </goal>
  </plan>
  </goal>

  <goal id="g04" ds="third_step">
  <plan operator="sequence">
    <goal id="g4c" ds="celling_made" ttf="2 "/>
    <goal id="g8a" ds="tiles_made" ttf="2 week"/>
    <goal id="g5c" ds="plumber_finition" ttf="4 week"/>
    <goal id="g6c" ds="electrician_finition" ttf="4 week"/>
    <goal id="g4d" ds="interior_painted" ttf="2 weeks"/>
  </plan>
  </goal>
 </plan>
</goal>
```

```
<mission id="manage_building" min="1" max="1">
  <adoptRequirementMission/>
  <leaveRequirementMission/>
  <goal id="g0"/>
</mission>
<mission id="prepare_site" min="1" max="1">
  <adoptRequirementMission/>
  <leaveRequirementMission/>
  <goal id="g1a" />
  <goal id="g1b" />
</mission>
<mission id="build_structure" min="1" max="1">
  <adoptRequirementMission/>
  <leaveRequirementMission/>
  <goal id="g2a" />
  <goal id="g2b" />
</mission>
<mission id="build_roof" min="1" max="1">
  <adoptRequirementMission/>
  <leaveRequirementMission/>
  <goal id="g3a" />
  <goal id="g3b" />
</mission>
<mission id="build_int_wall" min="1" max="1">
  <adoptRequirementMission/>
  <leaveRequirementMission/>
  <goal id="g4a" />
  <goal id="g4b" />
  <goal id="g4c" />
  <goal id="g4d" />
</mission>
<mission id="install_plumbing" min="1" max="1">
  <adoptRequirementMission>
    <adoptReq id="ar1m5" type="AdoptReq" property="mand" expression="has_standard == DTU 60.11"/>
    <adoptReq id="ar2m5" type="AdoptReq" property="mand" expression="has_standard == DTU 65"/>
  </adoptRequirementMission>
  <leaveRequirementMission/>
  <goal id="g5a" />
  <goal id="g5b" />
  <goal id="g5c" />
</mission>
  <mission id="install_electricity" min="1" max="1">
    <adoptRequirementMission>
      <adoptReq id="ar1m6" type="AdoptReq" property="mand" expression="has_standard == NF C 15-100"/>
    </adoptRequirementMission>
    <leaveRequirementMission/>
    <goal id="g6a" />
    <goal id="g6b" />
    <goal id="g6c" />
  </mission>
  <mission id="fit_door-windows" min="1" max="2">
    <adoptRequirementMission/>
    <leaveRequirementMission/>
    <goal id="g7a" />
    <goal id="g7b" />
  </mission>
```

```
    <mission id="fit_tiler" min="1" max="1">
      <adoptRequirementMission>
        <adoptReq id="ar1m8" type="AdoptReq" property="mand" expression="has_standard == DTU 52.1"/>
      </adoptRequirementMission>
      <leaveRequirementMission/>
      <goal id="g8a" />
    </mission>
    <mission id="fit_frontage" min="1" max="1">
      <adoptRequirementMission/>
      <leaveRequirementMission/>
      <goal id="g9a" />
    </mission>
  </scheme>
</functional-specification>

<normative-specification>
 <norm id="n0"  type="obligation"  role="house_owner" mission="manage_building"/>
 <norm id="n1"  type="obligation"  role="site_prep_contractor" mission="prepare_site"/>
 <norm id="n2"  type="obligation"  role="bricklayer"  mission="build_structure"/>
 <norm id="n3"  type="obligation"  role="roofer"      mission="build_roof"/>
 <norm id="n4"  type="obligation"  role="plasterer"   mission="build_int_wall"/>
 <norm id="n5"  type="obligation"  role="plumber"     mission="install_plumbing"/>
 <norm id="n6"  type="obligation"  role="electrician" mission="install_electricity"/>
 <norm id="n7"  type="obligation"  role="joiner"      mission="fit_door-windows"/>
 <norm id="n8"  type="obligation"  role="tiles_layer" mission="fit_tiler"/>
 <norm id="n9"  type="obligation"  role="frontage_maker" mission="fit_frontage"/>
</normative-specification>

<entryExit-specification>
  <portal id="p1">
    <entry-requirements>
      <entryReq id="enr1p1" type="EntryReq" property="mand" expression="turnover == _"/>
      <entryReq id="enr2p1" type="EntryReq" property="opt" expression="staff_size == _"/>
      <entryReq id="enr3p1" type="EntryReq" property="mand" expression="environment_rules == _"/>
    </entry-requirements>
    <exit-requirements/>
    <gates>
      <gate id="ga0" target="(house_owner, _, _)"
            groupId="building_grp" schemeId="build_house_sch" deadline="1mn"/>
      <gate id="ga1" target="(site_prep_contractor, _, _)"
            groupId="bigworks_grp" schemeId="build_house_sch" deadline="1weeks"/>
      <gate id="ga2" target="(bricklayer, _, _)"
            groupId="bigworks_grp" schemeId="build_house_sch" deadline="1weeks"/>
      <gate id="ga3" target="(roofer, _, _)"
            groupId="bigworks_grp" schemeId="build_house_sch" deadline="1weeks"/>
    </gates>
    <associated-groups>
      <associated-group id="assGr1" group="bigworks_grp" portalType="entryExit">
        <gate id="ga1"/>
        <gate id="ga2"/>
        <gate id="ga3"/>
      </associated-group>
      <associated-group id="assGr0" group="building_grp" portalType="entryExit">
        <gate id="ga0"/>
      </associated-group>
    </associated-groups>
```

```
  </portal>

  <portal id="p2" >
    <entry-requirements>
      <entryReq id="enr1p2" type="EntryReq" property="mand" expression="turnover == _"/>
      <entryReq id="enr2p2" type="EntryReq" property="mand" expression="staff_size &gt;= 15"/>
      <entryReq id="enr3p2" type="EntryReq" property="mand" expression="environment_rules == _"/>
    </entry-requirements>
    <exit-requirements/>
    <gates>
      <gate id="ga4" target="(plasterer, _, _)"
            groupId="fitting_grp" schemeId="build_house_sch" deadline="1weeks"/>
      <gate id="ga5" target="(plumber, _, _)"
            groupId="fitting_grp" schemeId="build_house_sch" deadline="1weeks"/>
      <gate id="ga6" target="(electrician, _, _)"
            groupId="fitting_grp" schemeId="build_house_sch" deadline="1weeks"/>
    </gates>
    <associated-groups>
      <associated-group id="assGr2" group="fitting_grp" portalType="entryExit">
        <gate id="ga4"/>
        <gate id="ga5"/>
        <gate id="ga6"/>
      </associated-group>
    </associated-groups>
  </portal>

  <portal id="p3">
    <entry-requirements>
      <entryReq id="enr1p3" type="EntryReq" property="mand" expression="3 &lt;= turnover"/>
      <entryReq id="enr2p3" type="EntryReq" property="mand" expression="staff_size == _"/>
      <entryReq id="enr3p3" type="EntryReq" property="mand" expression="environment_rules == _"/>
    </entry-requirements>
    <exit-requirements/>
    <gates>
      <gate id="ga7" target="(joiner, _, _)"
            groupId="finition_grp" schemeId="build_house_sch" deadline="1weeks"/>
      <gate id="ga8" target="(tiles_layer, m8, _)"
            groupId="finition_grp" schemeId="build_house_sch" deadline="1weeks"/>
      <gate id="ga9" target="(frontage_maker, m9, g9a)"
            groupId="fintion_grp" schemeId="build_house_sch" deadline="1weeks"/>
    </gates>
    <associated-groups>
      <associated-group id="assGr3" group="finition_grp" portalType="entryExit">
        <gate id="ga7"/>
        <gate id="ga8"/>
        <gate id="ga9"/>
      </associated-group>
    </associated-groups>
  </portal>
 </entryExit-specification>
</organisational-specification>
```

Annexe E

Expérimentations : Cas d'étude Build-House

Dans cette annexe, nous présentons quelques copies d'écrans des résultats des expérimentations réalisées pour la gestion d'une organisation multi-agent ouverte avec notre cas d'étude build-house.

E.1 Création des artefacts organisationnels

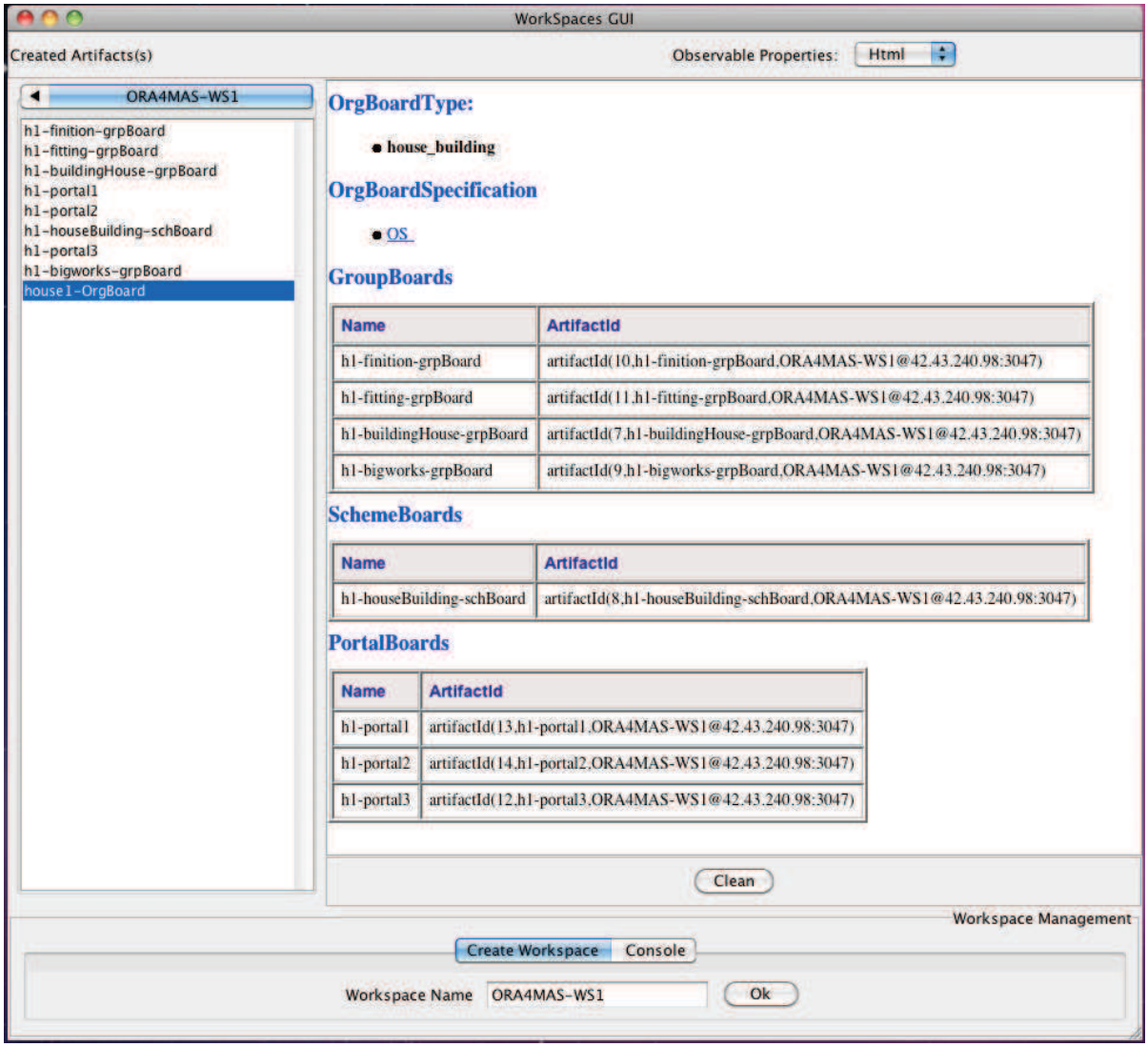


FIGURE E.1 – OrgBoard de l'entité organisationnelle : **house1-OrgBoard**

FIGURE E.2 – GroupBoard building_grp : **h1-buildingHouse-grpBoard**

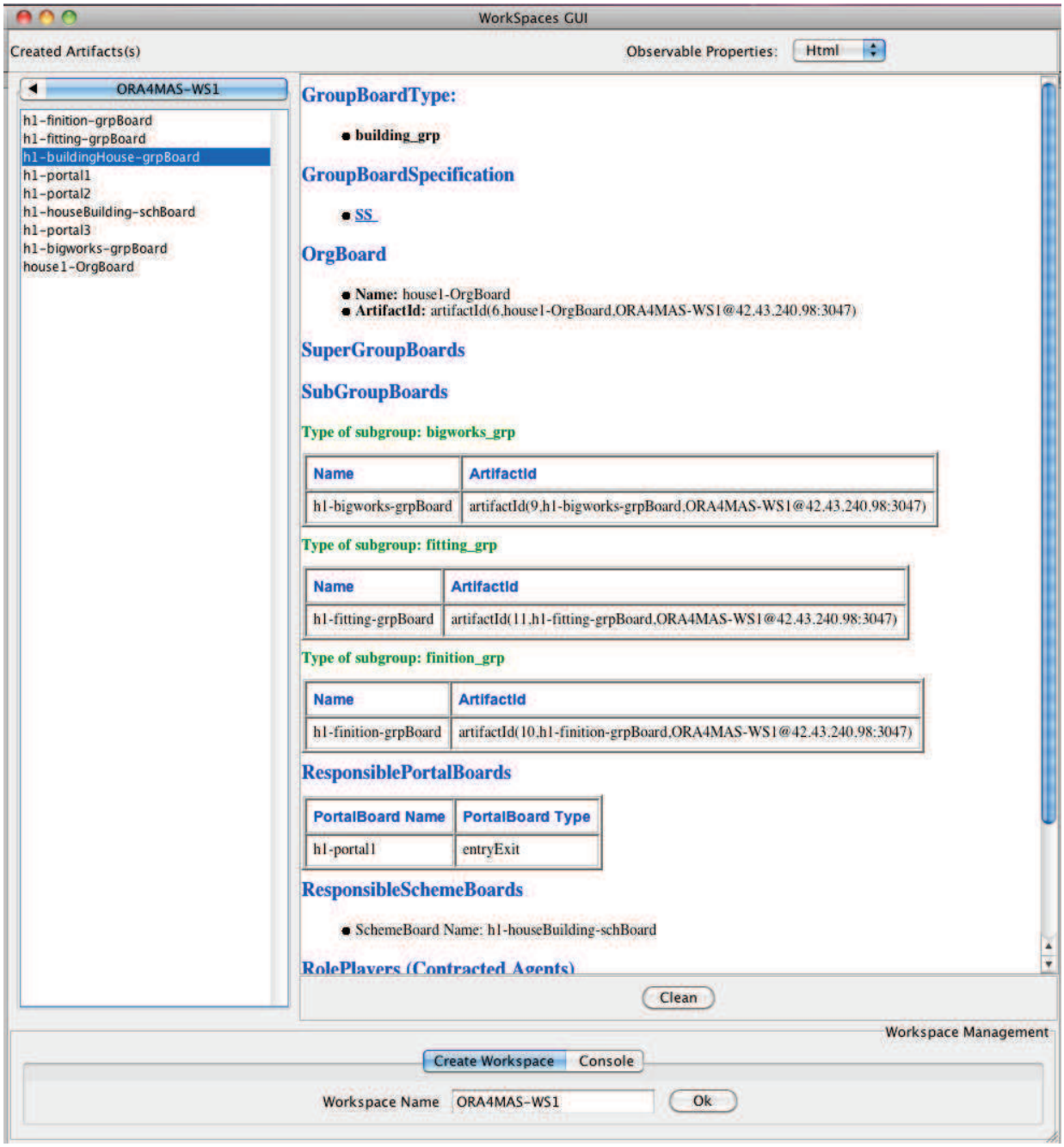


FIGURE E.3 – GroupBoard fitting_grp : **h1-fitting-grpBoard**

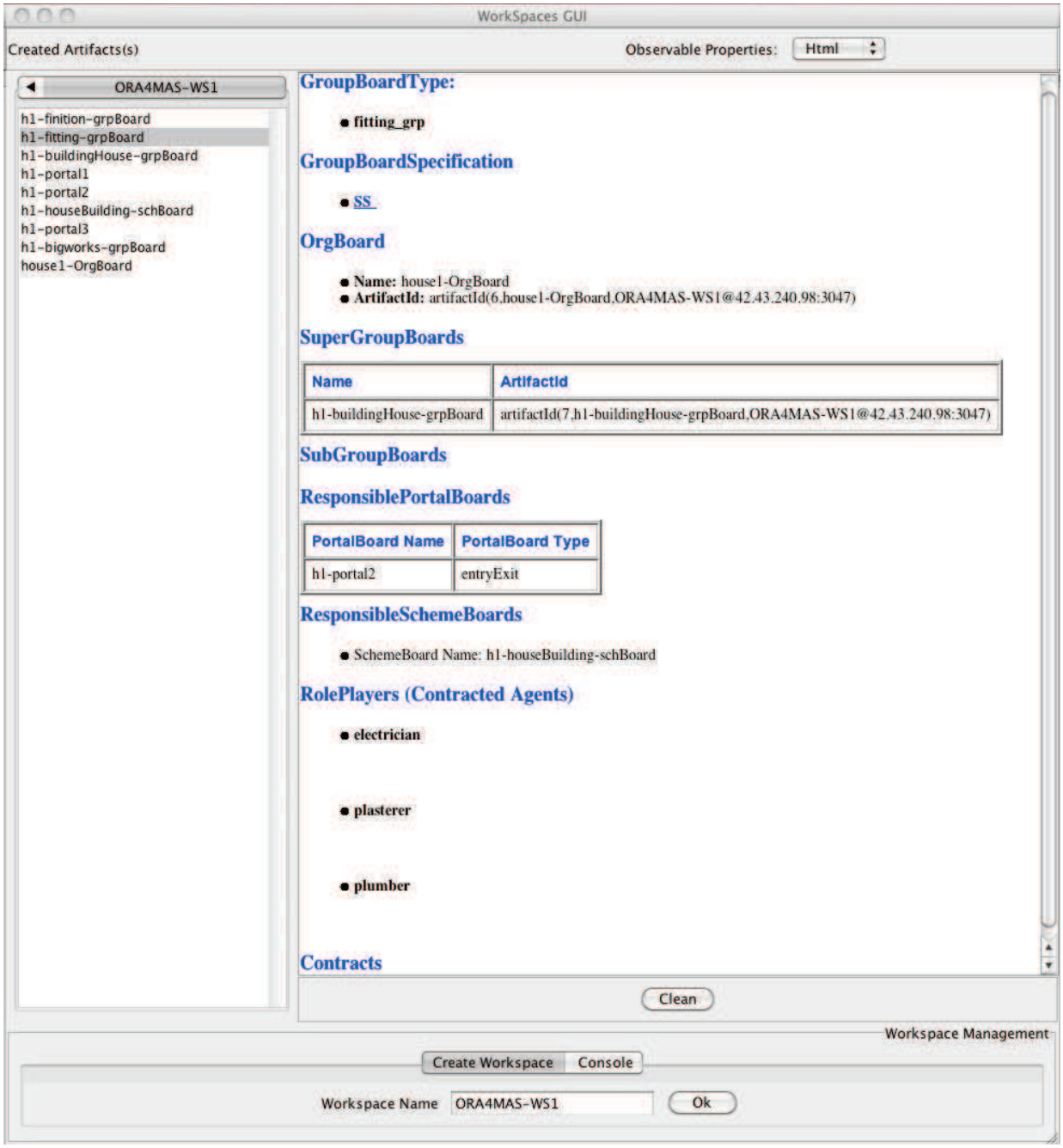


FIGURE E.4 – PortalBoard p2 : **h1-portal2** (1)

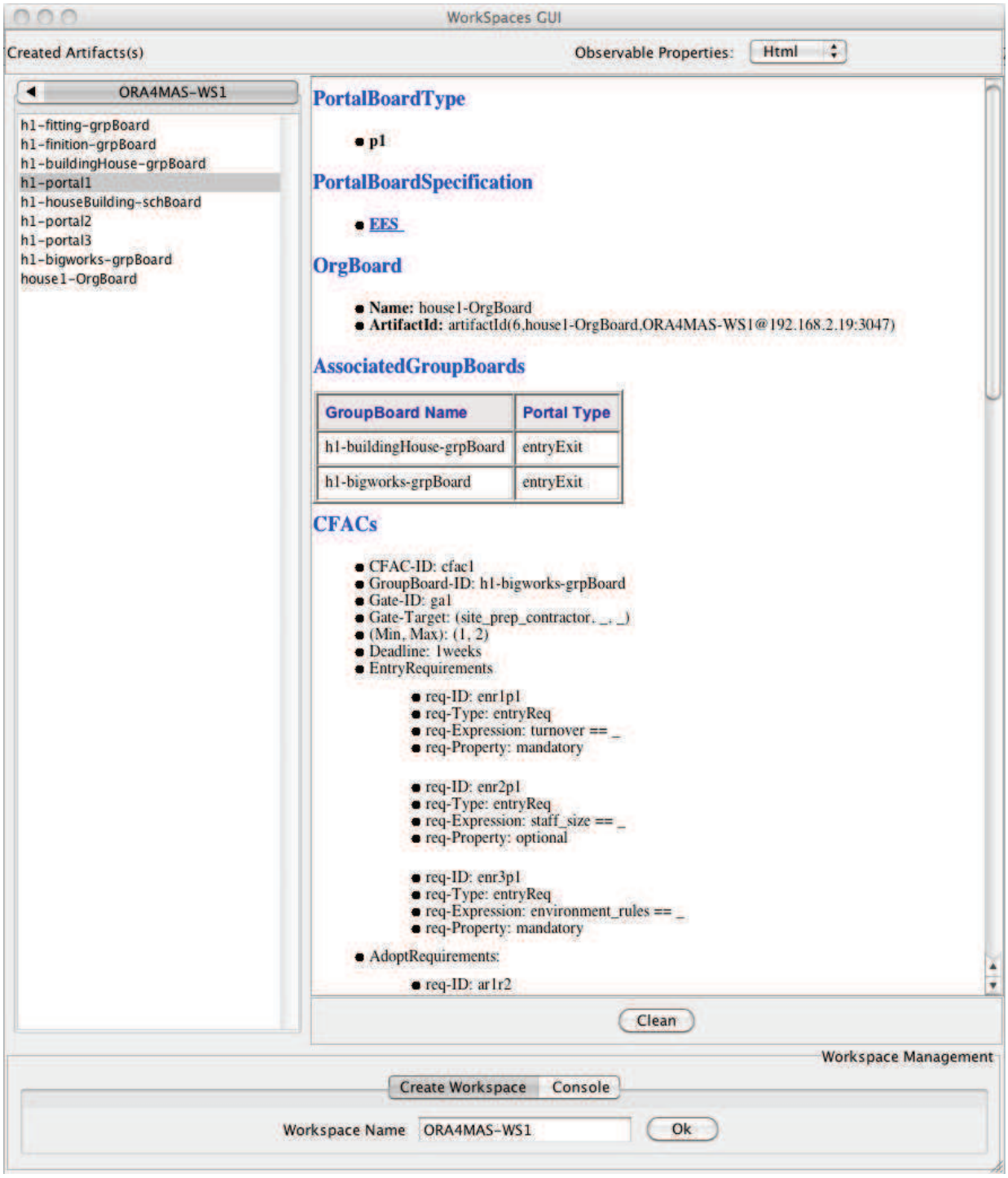


FIGURE E.5 – PortalBoard p2 : **h1-portal2** (2)

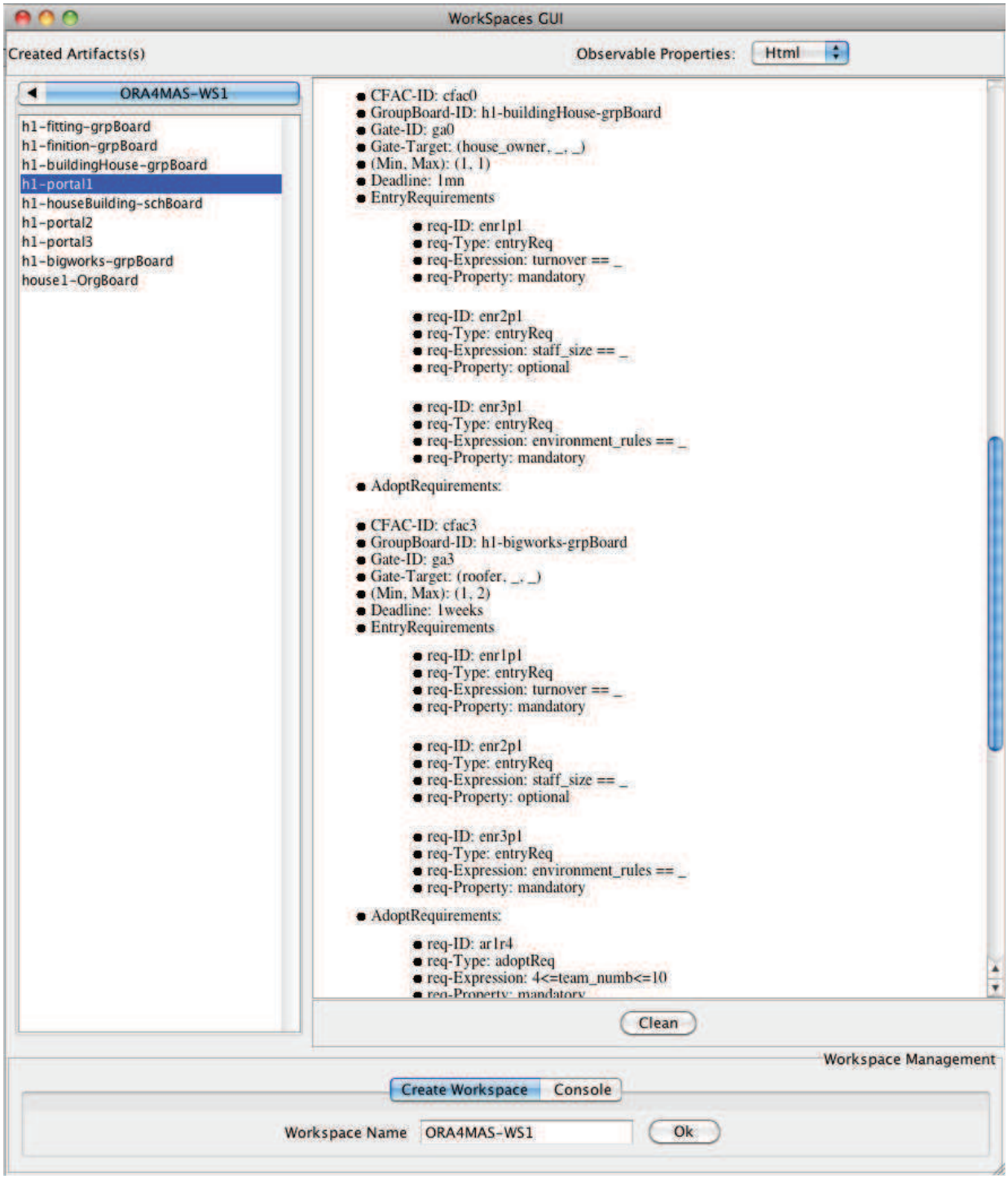


FIGURE E.6 – PortalBoard p3 : **h1-portal3** (1)

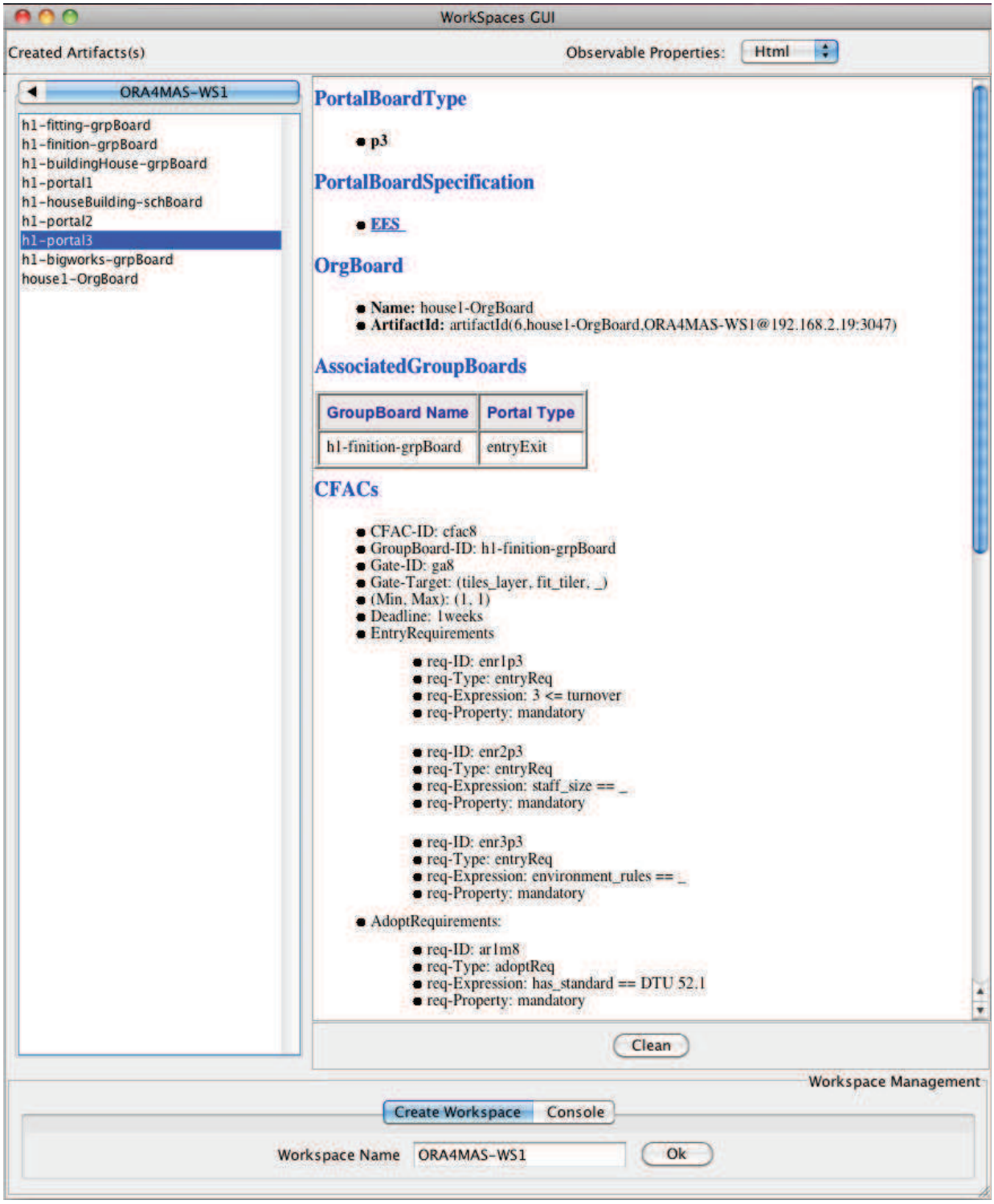


FIGURE E.7 – PortalBoard p3 : **h1-portal3 (2)**

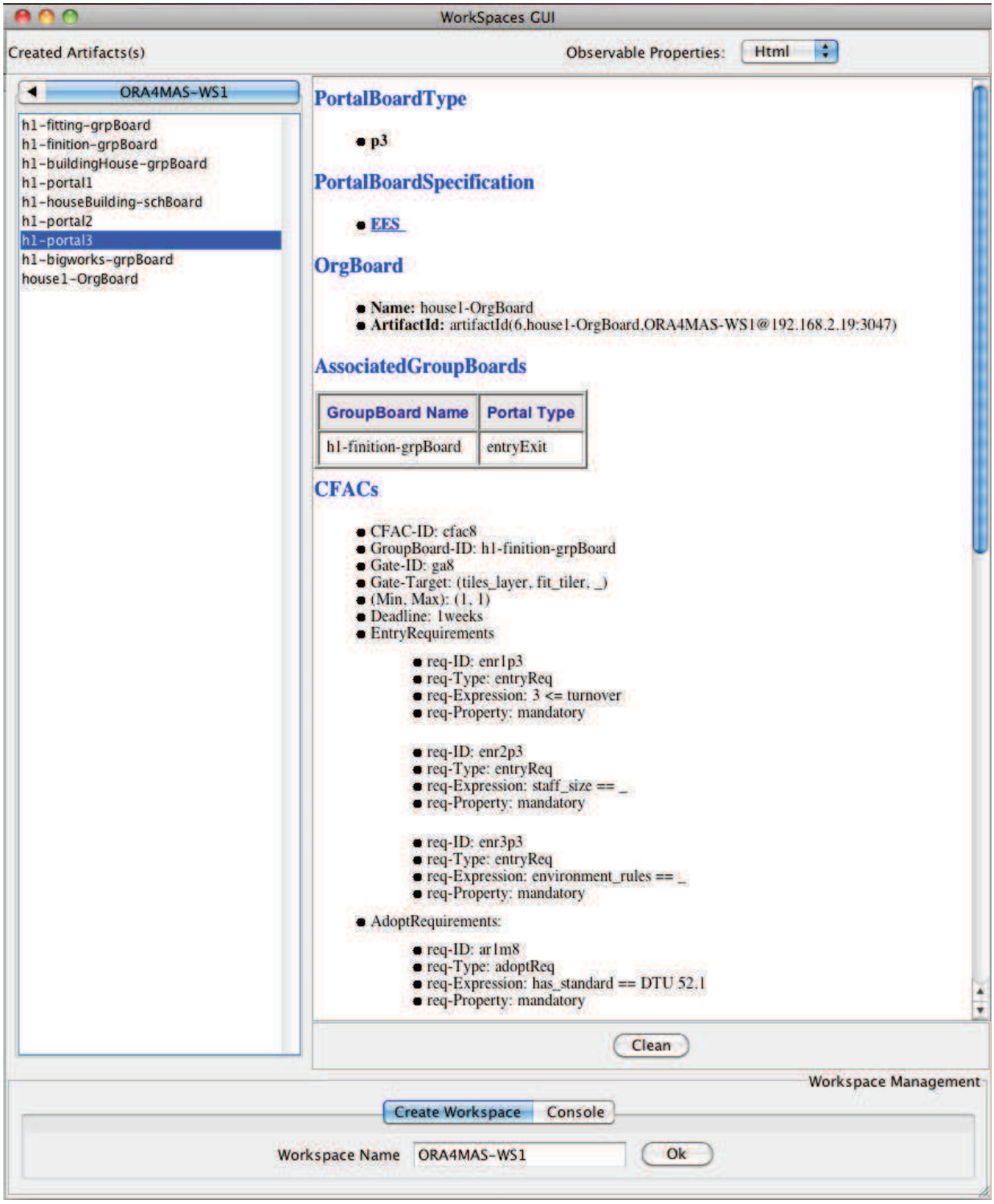
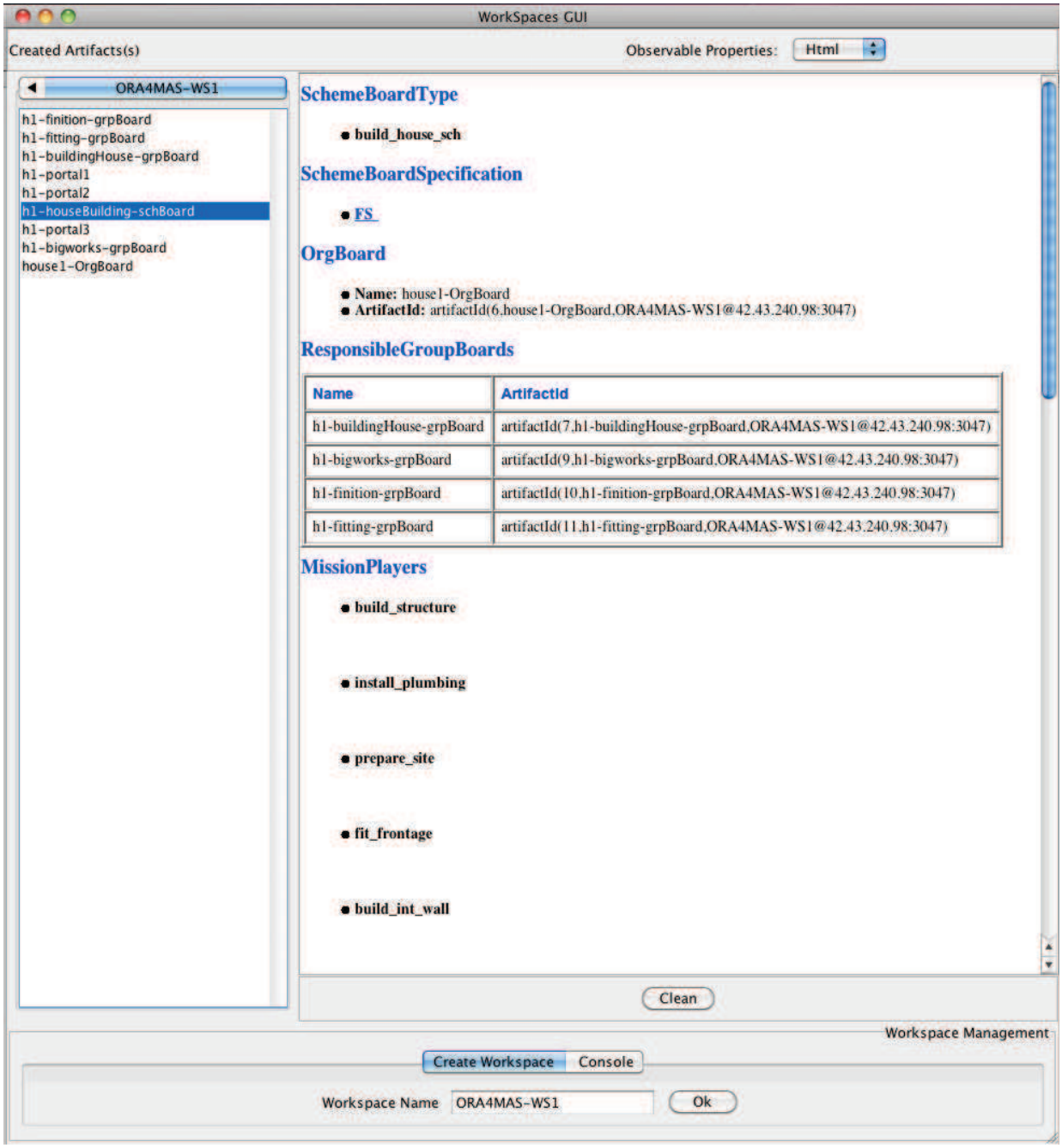


FIGURE E.8 – SchemeBoard de l'entité organisationnelle : **h1-houseBuilding-schBoard**



E.2 Gestion des procédures d'entrées

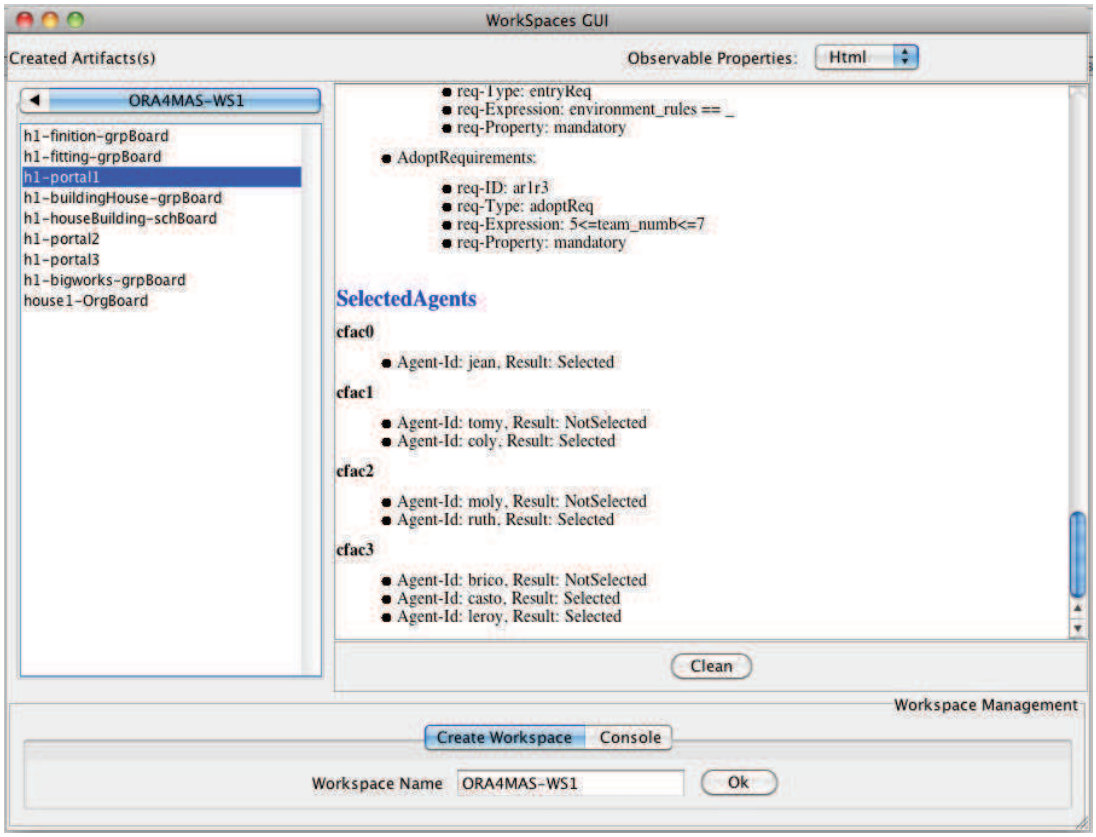
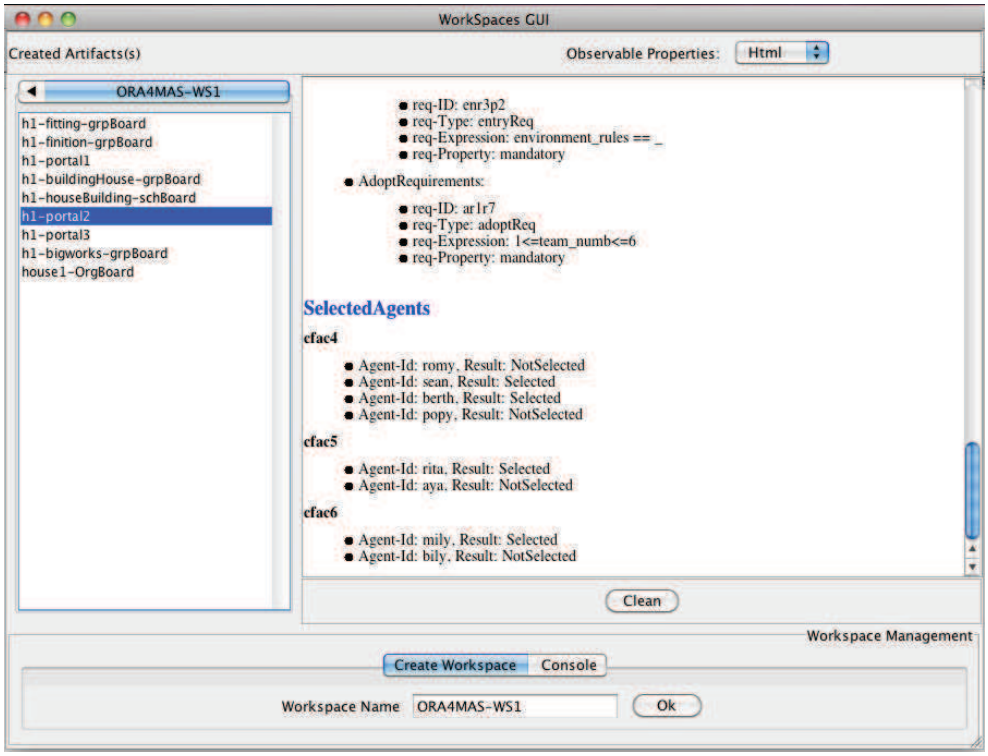
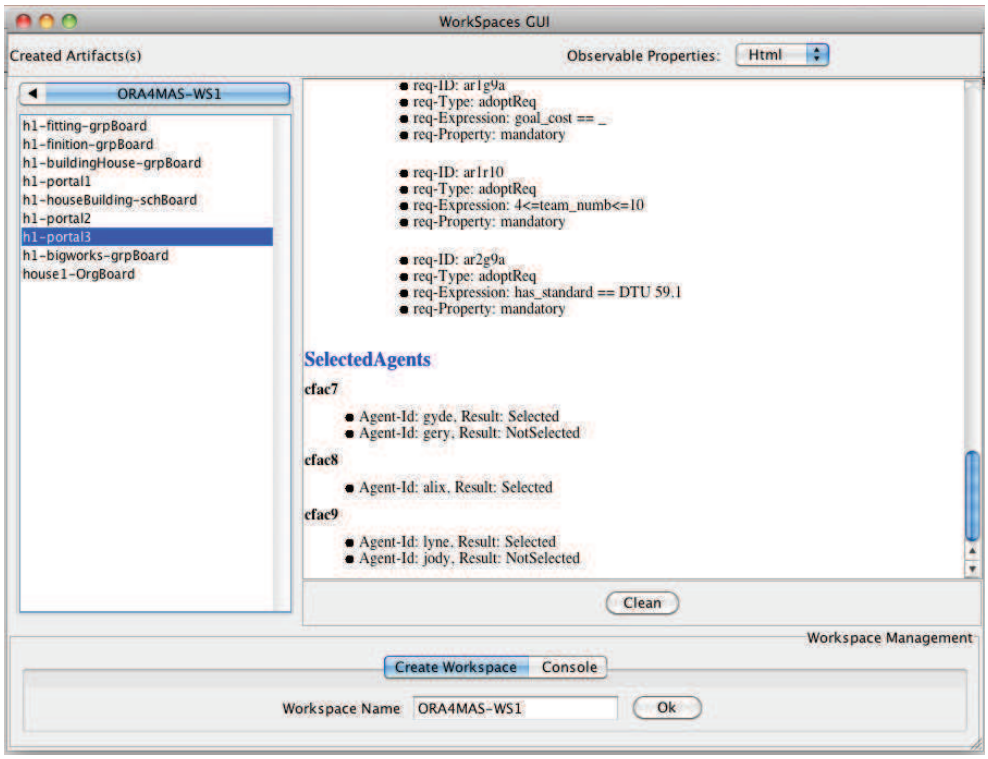


FIGURE E.9 – Propriétés observables SelectedAgent du portail 1



(a) Propriétés observables SelectedAgent du portail 2



(b) Propriétés observables SelectedAgent du portail 3

FIGURE E.10 – Gestion processus d'entrée par les PortalBoard

FIGURE E.11 – Gestion processus d'entrée : Contrats des agents *Leroy* et *Ruth*

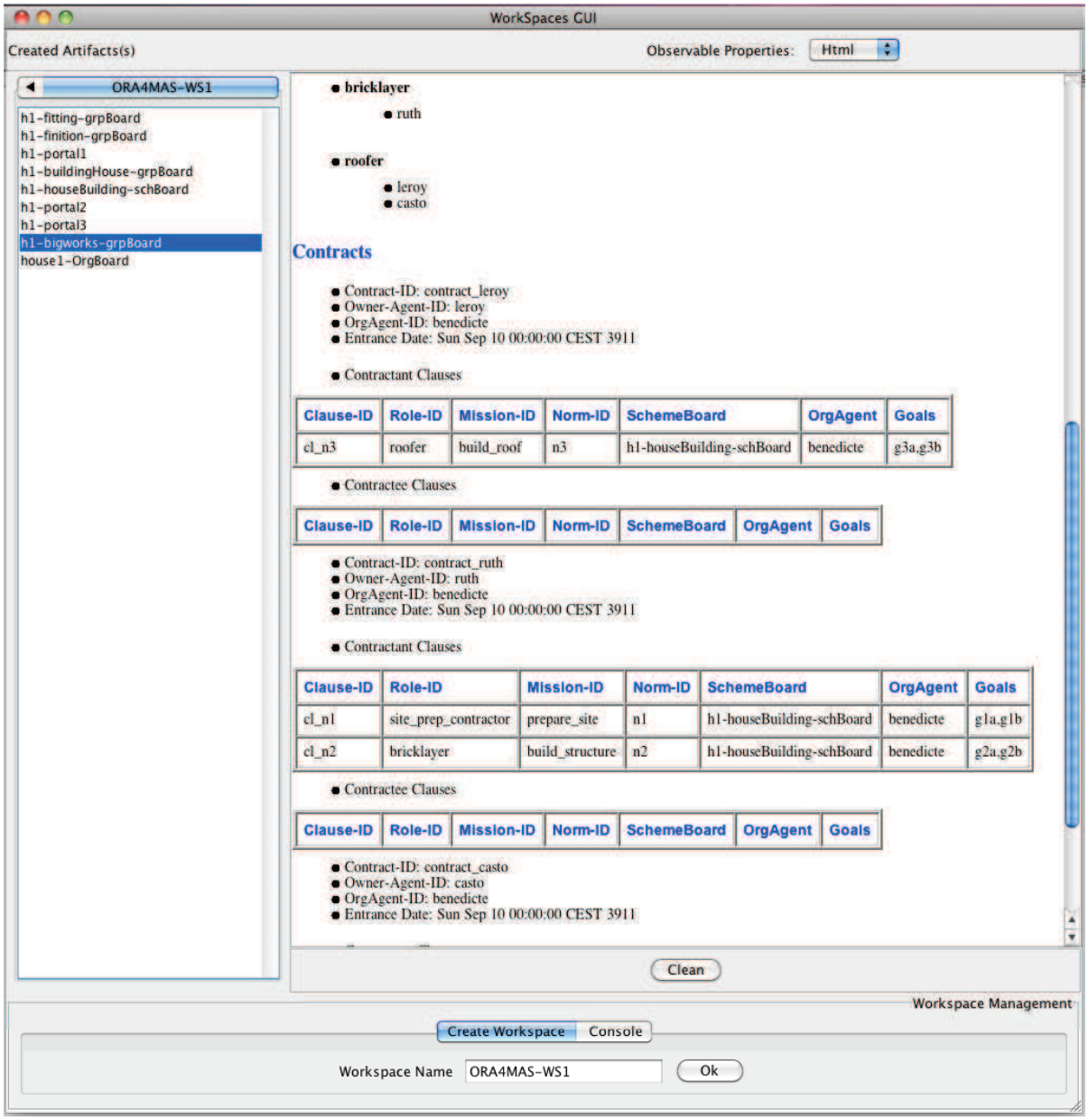
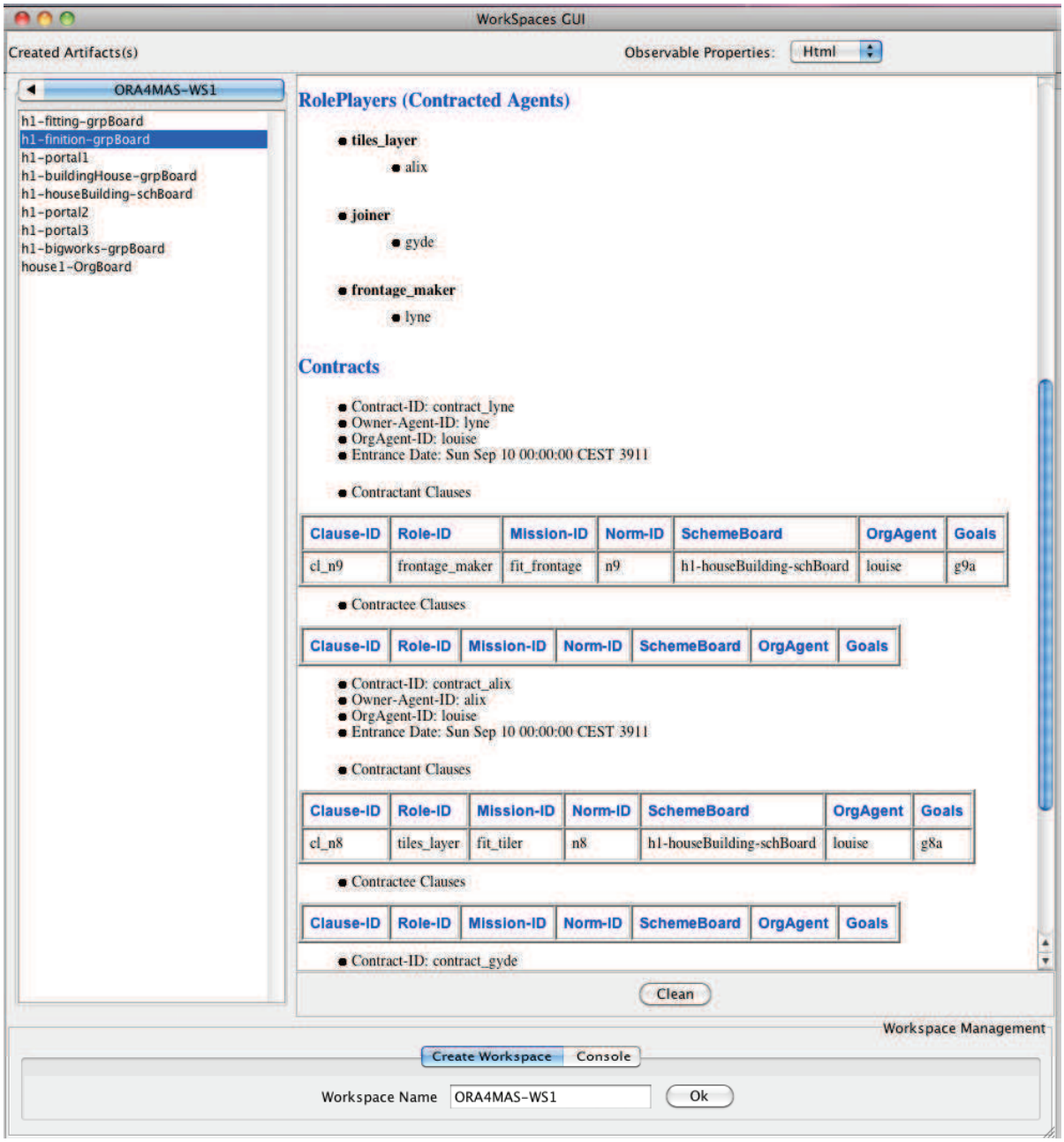


FIGURE E.12 – Gestion processus d'entrée : Contrats des agents *Lyne* et *Alix*



E.3 Gestion des engagements aux missions et de la régulation

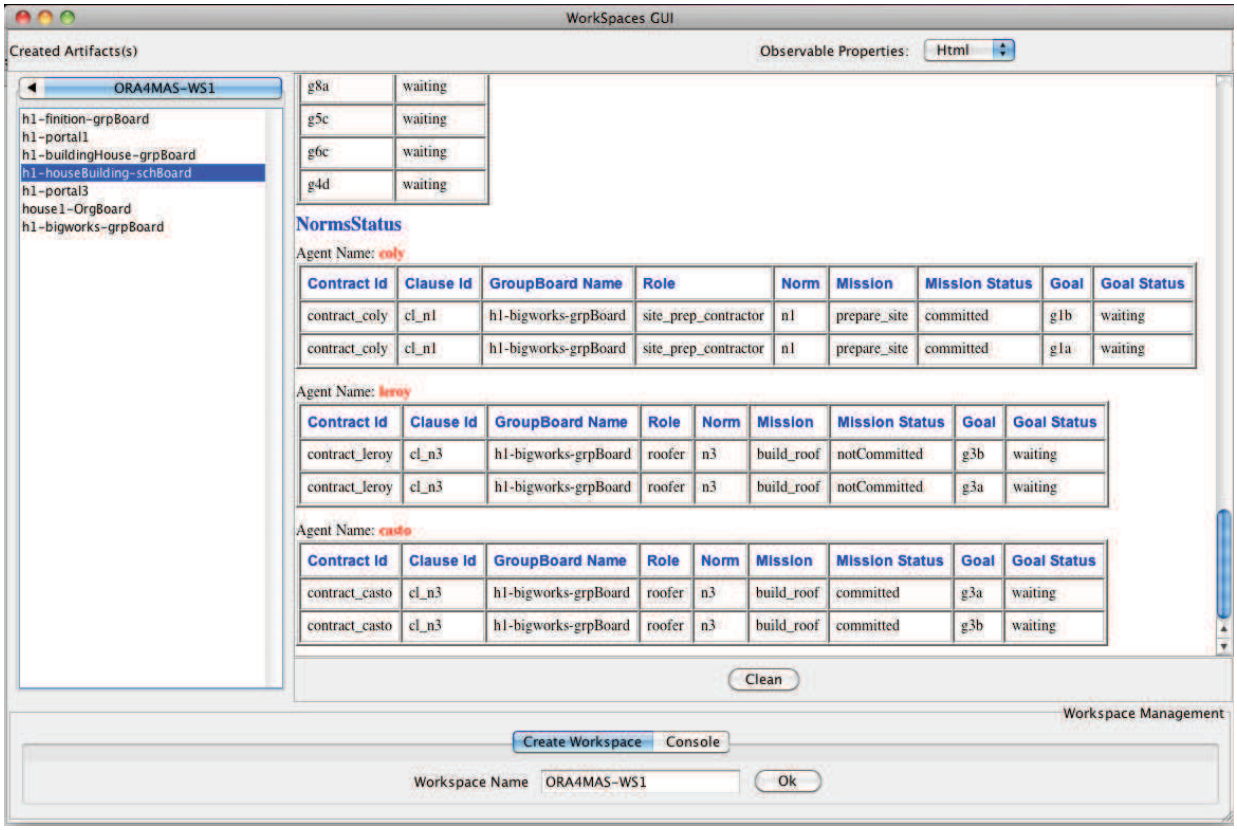


FIGURE E.13 – Engagement aux mission du schéma social

École Nationale Supérieure des Mines de Saint-Étienne

N° d'ordre : 2011 EMSE 0630

ROSINE KITIO TEUSSOP

GESTION DE L'OUVERTURE AU SEIN D'ORGANISATIONS MULTI-AGENTS
UNE APPROCHE BASÉE SUR DES ARTEFACTS ORGANISATIONNELS

Spécialité : Informatique

Mots Clefs : Organisation Multi-agent, Ouverture, Normes, Artefacts Organisationnels, Gestion entrée / sortie ...

Résumé : Les systèmes multi-agents sont des systèmes dans lesquels des entités logicielles appelées agents interagissent de façon autonome dans des environnements partagés. Ces dernières années, de nombreuses recherches sur les organisations multi-agents ont été menées et divers modèles organisationnels ont été proposés. Cependant, ils n'offrent pas de solution pour une gestion effective de la problématique d'ouverture dans des organisations multi-agents normatives. Dans cette thèse, nous nous sommes intéressées à l'étude de cette problématique et donc à la spécification des besoins relatifs à la mise en oeuvre de l'ouverture au sein d'organisation multi-agent. Nous avons ainsi identifié trois propriétés caractéristiques de cette problématique : l'interopérabilité d'une organisation avec son environnement extérieur et interne, la gestion des entrées / sorties et la gestion du contrôle et de la régulation des agents. Pour répondre à ces propriétés, nous avons proposé un langage de modélisation d'organisation (OML) MOISE qui est une extension de Moise+. MOISE permet de spécifier de façon explicite les processus d'entrée / sortie dans une organisation et notamment les exigences relatives aux missions, buts, et rôles de l'organisation. Nous avons également proposé une infrastructure de gestion d'organisation (OMI) ORA4MAS qui s'inspire du méta-modèle Agents et Artifacts (A&A). Nous avons défini le concept d'*artefact organisationnel* pour implémenter les fonctionnalités correspondant aux spécifications du langage MOISE. Nos propositions ont été illustrées avec une spécification d'organisation de gestion de la construction d'un édifice. La mise en oeuvre des propriétés d'ouverture a été expérimentée avec la gestion des processus d'entrée / sortie des agents, la négociation des clauses de contrat, la coordination des coopérations des agents à la réalisation des buts de construction d'un édifice, le contrôle des comportements des agents relativement aux normes de l'organisation ainsi que leur régulation.

École Nationale Supérieure des Mines de Saint-Étienne

N° d'ordre : 2011 EMSE 0630

ROSINE KITIO TEUSSOP

GESTION DE L'OUVERTURE AU SEIN D'ORGANISATIONS MULTI-AGENTS
UNE APPROCHE BASÉE SUR DES ARTEFACTS ORGANISATIONNELS

Major in Computer Science

Keywords : Multi-agent Organization, Open MAS, Organizational Artifacts, Entry / Exit Management.

Abstract : Multi-Agent Technology concerns the development of decentralized and open systems composed of different agents interacting in a shared environment. In recent years, organization has become an important in this research field. Many models have been, and are still, proposed. While no concensual model emerges of these different works, it appears that they all lack the ability to build open and normative organizations in the sense of management of entry / exit of agents into organization but also decentralized control / regulation of the autonomy of the agents. In this thesis, our objective consists in the definition of a new model addressing these requirements. Ours reseaches allow us to extend the $\mathcal{M}$OISE$^+$ organizational modeling language (OML) in a new version namming MOISE. In this one we define an Entry / Exit specification allowing to explicitly specify the ways in which the agents can enter or exit in or from an organisation by providing some requirements according to the missions, the goals and the roles of the organisation. The organizational management infrastructure (OMI) ORA4MAS proposed take advantage of the Agents and Artifacts (A&A) approach. We defined the *Organizational Artifacts* concept as the basic building block of our OMI for the management of organized and open MAS. To focus our study, the organizational artifacts will be defined considering the OML specification of the MOISE model. We experimented our proposal with the specification of an application aiming to manage the build of a house. We then experimented the management of the candidate agents to enter in the organisation and cooperate with the other to build the house according to a specified social scheme, the specified norms and their contract clauses negociated when they will be admitted in the organisation.